Linearization in μCRL

Yaroslav S. Usenko

Linearization in μCRL

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Usenko, Yaroslav S.

Linearization in muCRL / by Yaroslav S. Usenko. - Eindhoven : Technische Universiteit Eindhoven, 2002.
Proefschrift. - ISBN 90-386-0612-5
NUR 993
Subject headings: software verification / process algebra / program translators / programming languages ; data types / algebraic programming languages / programming ; formal methods / programming languages ; muCRL
CR Subject Classification (1998): D.2.4, I.1.2, F.3.1, D.3.3, D.3.1, I.1.1.

Typeset with LaTeX 2_ε
IPA Dissertation Series 2002-16
Print: Universiteitsdrukkerij, Technische Universiteit Eindhoven

The work in this thesis has been carried out under the auspices of the Institute for Programming Research and Algorithmics (IPA), at the Centre for Mathematics and Computer Science (CWI) in Amsterdam.

Linearization in μCRL

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 2 december 2002 om 16.00 uur

door

Yaroslav Sergiyovych Usenko

geboren te Kiev, Oekraïne

Dit proefschrift is goedgekeurd door de promotoren:
prof.dr.ir. J.F. Groote
en
prof.dr. W.J. Fokkink

Contents

Preface

The story that leads to this thesis began back in May 1996, when Kees Middelburg visited Kiev with lectures on process algebra and SDL semantics, and invited me to work on these subjects at UNU/IIST in Macau. There I learned that formal methods, specification and verification of software, were not just theory, but were actually used in reality. In particular, SDL provided a higher-level methodology for developing concurrent systems and communication protocols, and process algebra was a formal framework to manipulate with descriptions of such systems in a simple way, similar to transforming polynomial or trigonometric expressions. The meaning of process algebra expressions was defined on the level of graphs that represent behaviors on a lowest possible level of abstraction.

The result of that project (cf. [17]) was a semantics of SDL in process algebra, which helped to understand SDL specifications and made a formal analysis foreseeable. It became apparent that data influence the behavioral aspects of a system, so an extension of process algebra with a symbolic treatment of data would be a better semantic domain. A natural candidate was μCRL, a language based on process algebra and abstract data types, which had just been extended with an explicit treatment of time (cf. [47]).

A procedure to generate the symbolic representations of behavior graphs for μCRL expressions was still needed. Filling this gap was a goal when I came to the group of Jan Friso Groote at CWI in September 1998, and it became the research topic presented in this thesis.

During my work at CWI I participated in EU Project DR-TESY, in which ways to couple μCRL with other formal techniques like statecharts and an SDL-like formalism were investigated (cf. [11]). In another project, initiated by Royal Dutch Navy, a combination of μCRL and B was studied (cf. [40]). Both of these projects had to do with using μCRL for the analysis of higher-level specification formalisms. I also worked on a specification of the HAVi leader election protocol in both μCRL and Spin, and on a translation method from μCRL to Spin (cf. [97]), which shows some peculiar differences between algebraic and imperative concurrency.

Acknowledgments

I would like to thank Kees Middelburg for introducing me to scientific research on the highest international level, and for taking care of me afterwards. I am also grateful

to him and his family for providing me with a place to stay during my first visits to Holland. My first promotor Jan Friso Groote has inspired me to work on software verification in general and on this thesis in particular. Moreover, he appears to be an excellent manager who always helped me to put things back on track. Jan Bergstra was always willing to discuss and give an advice on basically any topic, be it a research one, or not. During his visit to Macau he helped me in finishing the project and strengthened my belief in continuing with scientific research. Alban Ponse, was a coauthor of two of the papers this thesis is based upon. Writing those papers with him was an experience full of interesting discussions, challenging problems, and fun. Wan Fokkink, my second promotor and second boss at CWI, read most of my papers, provided comments and helped to correct my English and style. I spent many days and evenings sharing the office with Vincent van Oostrom, which I enjoyed both socially and scientifically. He brought me to the football team where I played weekly for more than three years, and he translated the summary of this thesis to Dutch.

My colleagues Bahareh Badban, Stephan Blom, Conrado Daws Poulastrou, Natalia Ioustinova, Izak van Langevelde, Bert Lisser, Bas Luttik, Sjouke Mauw, Simona Orzan, Jun Pang, Jaco van de Pol, Michel Reniers, Judi Romijn, Miguel Valero Espada, Jos van Wamel, and Mark van der Zwaag, made the group SEN2 at CWI a great working environment. I am grateful to Holger Hermanns and other colleagues at the FMT group, University of Twente, for providing me with additional inspiration for finishing this thesis.

This thesis would not be possible without a solid computer science education I received from Taras Shevchenko University of Kyiv, especially from Pavlo Gorshkov, Alexander Letichevsky, Leonid Lisovik, Mykola Nikitchenko and Volodymyr Red'ko. At UNU/IIST I enjoyed the lectures by Dines Bjørner, Zhou Chaochen, Chris George and Tomasz Janowski. At RISC-Linz I benefited from the courses by Bruno Buchberger, who improved my understanding of formal and informal reasoning.

I am grateful to the members of the promotion committee—Jos Baeten, Jan Bergstra, Wan Fokkink, Jan Friso Groote, Kees van Hee, Holger Hermanns, Kees Middelburg and Faron Moller—for finding time to read this thesis, and providing suggestions for improvements.

I thank my father for his strong, continuous, and persuasive encouragement in working on this thesis, and the other members of my family for backing this encouragement and believing in me. Many of my friends, no matter how far they were, helped me with words and deeds. I also thank Yuliya for her support, patience, and understanding throughout all these years.

Diemen, October 2002.

Chapter 1

Introduction

In this thesis we address the issue of linearization of recursive specifications in the specification language μCRL (micro Common Representation Language, [53, 47]) and in timed μCRL [47, 55]–an extension of the language to deal with time.

1.1 The Language μCRL

The language μCRL has been developed under the assumption that an extensive and mathematically precise study of the basic constructs of specification languages is fundamental to an analytical approach of much richer (and more complicated) specification languages such as SDL [104], LOTOS [66], PSF [73, 74] and CRL [92]. Moreover, it is assumed that μCRL and its proof theory provide a solid basis for the design and construction of tools for analysis and manipulation of distributed systems. The major design objectives (as stated in [47]) for μCRL were that:

- μCRL had to be so expressive that 'real life systems', generally consisting of a set of interacting programs, could be described;
- μCRL had to be so simple and clear that it was suitable as a basis for mathematical analysis; and
- the definition of μCRL had to be sufficiently precise to allow for the independent construction of computer tools for μCRL, capable of assisting in the actual development of systems.

μCRL was successfully applied in the analysis of a wide range of protocols and distributed systems. The number of case studies performed in the language show that these design objectives were met. A large number of case studies is mentioned in [47]. Recently μCRL was used to support the optimized redesign of the Transactions Capabilities Procedures in the SS No. 7 protocol stack for telephone exchanges [5], to detect a number of mistakes in an industrial protocol over the CAN bus for lifting trucks [51] and in the Needham-Schroeder public-key protocol [80], and to analyze the coordination languages SPLICE [37, 64] and JavaSpaces [85].

The language μCRL offers a uniform framework for the specification of data and processes. Data are specified by equational specifications (cf. [14, 68]): one can declare sorts and functions working upon these sorts, and describe the meaning of these functions by equational axioms. Processes are described in process algebraic style, where the particular process syntax stems from ACP [15, 10, 39], extended with data-parametric ingredients: there are constructs for conditional composition, and for data-parametric choice and communication. As is common in process algebra, infinite processes are specified by means of (finite systems of) recursive equations. In μCRL such equations can also be data-parametric. As an example, for action a and adopting standard semantics for μCRL, each solution for the equation $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ specifies (or "identifies") the process that can only repeatedly execute a, and so does $\mathsf{Y}(17)$ where $\mathsf{Y}(n)$ is defined by the data-parametric equation $\mathsf{Y}(n) = \mathsf{a} \cdot \mathsf{Y}(n+1)$ with $n \in \mathit{Nat}$.

Similar to many process theories, including the value passing theories based on CCS (cf. [65]), the standard semantics of μCRL (cf. [53]) is based on Labeled Transition Systems (LTS), defined using Structured Operational Semantics (SOS) rules (cf. [2]). Labeled Transition Systems are directed graphs with the arcs labeled by actions. Many equivalence relations have been defined for LTSs (cf. [45, 44]), which gives a possibility to analyze whether two specifications in μCRL are equivalent. Model checking techniques (cf. [29, 71, 93]) have also been developed for LTSs, especially for the finite ones, which gives a possibility to prove properties of μCRL specifications. Many imperative concurrent languages have an LTS semantics (cf [63]) as well.

For most real-life examples, however, the underlying LTSs are extremely large or infinite. This brings the need for a symbolic representation, from which the LTSs could be generated in a relatively simple way. Despite the fact that symbolic techniques have been developed for analyzing LTSs (cf. [65, 25]), there is no commonly used format for defining an LTS in a symbolic, syntactic way.

In the setting of μCRL such a representation is called a Linear Process Equation (LPE). This is a restricted form of a μCRL equation, which is similar to a right-linear or GNF grammar used in language theory. Having a system description in the LPE format means that there is a constructive way of exploring its behavior by generating the LTS, or by applying symbolic analysis techniques.

1.2 Linear Process Equations

Linear Process Equations are an interesting subclass of systems of recursive equations, which contain only one *linear* equation, as defined on the next page. Here, linearity refers both to the form of recursion allowed, and to a restriction on the process operations allowed. The above examples $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ and $\mathsf{Y}(n) = \mathsf{a} \cdot \mathsf{Y}(n+1)$ are both LPEs. The restriction to LPE format still yields an expressive setting (for example, it is not hard to show that each computable process over a finite set of actions can be simply defined using an LPE containing only computable functions over the natural numbers, cf. [87]). Moreover, in the design and construction of tools for μCRL, LPEs establish a basic and convenient format, that can be seen as a symbolic representation of LTSs. This applies, for example, to tools for the generation of transition systems, or tools for optimization, deadlock checking, or simulation [21], all of which are based on term rewriting. However, the real potential of the LPE format is in symbolic techniques that enable the analysis of large or even infinite systems. Some of these are based on an equational theorem prover [83], invariants [20], the "cones and foci" method [57], or confluence reduction [22].

The LPE format stems from [20], in which the notion of a *process operator* is distinguished, and a proof technique for dealing with convergent LPEs is defined. There is a strong resemblance between LPEs and specifications in UNITY [27, 24], I/O automata [70], and a special case of recursive applicative program schemes [31, 32]. The restriction to linear systems has a long tradition in process algebra. For instance, restricting to so-called linear *specifications*, i.e., linear systems that in some distinguished model have a unique solution per variable, various completeness results were proved in a simple fashion (cf. [77, 16]). However, without data-parametric constructs for process specification, the expressiveness is limited: only regular processes can be defined (cf. [39, page 40]).

A Linear Process Equation has the following form [1]:

$$\begin{aligned}\mathsf{X}(d{:}D) = &\sum_{i\in I}\sum_{e_i:E_i} \mathsf{a}_i(f_i(d,e_i))\cdot \mathsf{X}(g_i(d,e_i)) \lhd c_i(d,e_i) \rhd \delta \\ &+\sum_{j\in J}\sum_{e_j:E_j} \mathsf{a}_j(f_j(d,e_j)) \lhd c_j(d,e_j) \rhd \delta\end{aligned}$$

where I and J are disjoint finite sets of indexes. Normally we are interested in a solution of the LPE in a particular initial state t_0. The equation is explained as follows. The process X being in a state d can, for any e_i that satisfy the condition $c_i(d,e_i)$, perform an action a_i parameterized by $f_i(d,e_i)$, and then proceed to the state $g_i(d,e_i)$. Moreover, it can, for any e_j that satisfy the condition $c_j(d,e_j)$, perform an action a_j parameterized by $f_j(d,e_j)$, and then terminate successfully.

Several symbolic techniques have been developed for LPEs. In [20] invariants have been defined for LPEs. An invariant is a boolean formula $I : D \to Bool$ such that it holds in the initial state ($I(t_0) \approx \mathbf{t}$) and if it holds in a state d and for some e_i the condition $c_i(d,e_i)$ is satisfied, then it also holds in the state $g_i(d,e_i)$. It is shown in [20] that in all models where all convergent LPEs have unique solutions, adding an invariant to a condition gives us an equivalent LPE. As this can make conditions more often equal to "false", the underlying LTS of the LPE can be significantly reduced.

Another possibility for LPE optimization is parameter space reduction. Some of the LPE parameters may not matter at all for the process behavior, or they may not change when the process goes from state to state. In some cases it is required to find an invariant or to split a compound parameter into a number of simpler ones in order to find the "dummy" parameters. Some of such parameter elimination techniques are described in [48]. In general, these optimizations are special cases of applying abstract interpretation (cf. [33, 35]) to the state space of the LPE.

In [22] the confluence for μCRL processes, defined in [56], is used to optimize LPEs in terms of the size of the underlying LTS. Intuitively, the different orders of executing ("silent") actions in distinct concurrent components usually end up in the

[1] For the precise definition, with vectors of parameters, we refer to Section 4.1.1.

same state. With the help of invariants, a number of transitions are identified as confluent and are removed from the LPE. This method is similar to partial order reduction methods [82].

A symbolic method for proving branching bisimilarity [44] of LPEs, called the cones and foci method, has been proposed in [57] and later extended to the timed μCRL setting in [103]. This method reduces the above problem to finding invariants and proving a number of first-order formulas about the data used in the state vector of the LPE.

Finally, in [50] a variant of modal μ-calculus for expressing properties of LPEs is presented. A symbolic model checker for this calculus is currently under development. The approach taken there is to symbolically reduce both the temporal formula and the LPE in a way that preserves the validity of the formula.

In case the symbolic techniques are not of help, analysis of LPEs can be done by explicitly exploring the entire reachable state space. This process leads to the generation of an explicit LTS and is described in [36]. So far, most of the successful case studies performed with the μCRL Toolset used explicit LTS analysis. Many tools for the analysis of distributed systems and communication protocols, like Spin [63] and CADP [38], are mainly using this explicit analysis method. In Spin this is possibly due to the fact that transforming imperative specifications is harder than the transforming algebraic specifications (for instance, term rewriting can be applied for the latter).

Most of the LPE verification techniques mentioned in this section (and many others) have been implemented in the μCRL Toolset [102]. Before these techniques can be applied, the μCRL specification under scrutiny has to be transformed to a "conditionally equivalent" LPE. Such a transformation procedure we call *linearization*. This is the topic of this thesis.

1.3 Linearization Problem

The language μCRL is considered to be a specification language because it contains ingredients that facilitate in a straightforward, natural way the modeling of distributed, communicating processes. In particular, it contains constructs for *parallelism*, *encapsulation* and *abstraction*. On the other hand, as mentioned above, LPEs constitute a basic fragment of μCRL in terms of expressiveness and tool support. This explains our interest in transforming any system of μCRL equations into an equivalent LPE, i.e., our interest to *linearize* μCRL process definitions.

In this thesis we present three linearization procedures.

- The first one (cf. Chapter 4) deals with a subset of μCRL called parallel pCRL [2]. In [20], pCRL (pico CRL) was defined as a fragment of μCRL. Essentially, pCRL restricts μCRL to the basic operations of process algebra, with data parametric choice, sequential composition and conditionals. Typically, in an LPE only pCRL syntax occurs. *Parallel* pCRL is an extension of pCRL in which a restricted use of more involved operations, such as $\|$ (parallel composition), is allowed. For example, in parallel pCRL the $\|$ may not occur within the scope

[2] A very similar linearization procedure (for parallel pCRL) is currently implemented in the μCRL Toolset.

of recursion. Very often distributed processes have a straightforward definition in parallel pCRL.

- The second procedure (cf. Chapter 5) extends the first one to the settings of full μCRL. The adaptation is rather straightforward, except for the last step, where a special data type is needed to model combinations of parallel and sequential compositions of processes.
- The third procedure (cf. Chapter 6) is an extension to the timed μCRL setting, where the equational theory of timed μCRL (cf. [55]) is extended and heavily exploited. The main complication of this case is in preserving well-timedness of the specification. In this setting the result of the linearization is called Timed LPE (TLPE), which could be "approximated" by untimed LPEs, so that the existing untimed tools can be applied to analyze timed systems.

We define the linearization algorithms on an abstract level, but in a very detailed manner. We do not concern ourselves with the question if and in what way systems of recursive equations define processes as their unique solutions (per variable). Instead, we argue that the transformation is correct in a more general sense: we show that linearization "preserves all solutions". This means that if a particular μCRL system of recursive equations defines a series of solutions for its variables in some model, then the LPE resulting from linearization has (at least) the same solutions for the associated process terms. Consequently, if the resulting LPE is such that one can infer that these solutions are *unique* in some particular (process) model, and the initial LPE has a solution in this model, then both systems define the same processes in that model. In our algorithms, most transformation steps satisfy a stronger property: the set of solutions is the same before and after the transformation. The presented linearization algorithms are developed with two additional goals in mind. We try to keep them optimal in terms of the size of generated LPE, briefly mentioning additional optimizations that could be applied. We also try to preserve the structure and the names of the initial specification as much as possible.

1.4 Equivalence of Systems of Equations

In process algebra, infinite behavior is usually specified by means of recursive equations.[3] We have already mentioned the simple example $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$, modeling a process that repeatedly executes action a. It is often convenient to consider a *system* of interdependent recursive equations. For instance, a communication protocol can be specified such that each of its parallel components (sender, receiver, etc.) is modeled by one or more equations. In the following we use the terminology 'system of recursive equations' to denote a set of one or more equations in the sense sketched above.

Although the specification of processes by means of systems of recursive equations serves its purpose well, a proof theory for this type of specification is not entirely trivial, and is equipped with various particular ingredients. For instance, we often

[3] An alternative method of specification is the use of recursive operations, such as the Kleene star [13], or the use of fixpoint operators [78].

want to assert that such a system represents a particular process in some intended model as the *unique* solution for one of its variables. As an example, the recursive equation $\mathsf{X} = \mathsf{X}$ has (in any model) any process as its solution, and the equation $\mathsf{X} = \mathsf{X} + \mathsf{a}$ (where $+$ models choice) has many solutions (in many models), whereas $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ has no solution in models that represent only finite processes. In the case that a system of recursive equations has a unique solution (per variable) in some intended model, we say that this system is a recursive *specification*: some intended process is specified by means of recursive equations. Often, establishing the uniqueness of solutions is intertwined with verification purposes. If one can show that each solution for some distinguished variable in a system of recursive equations is also a solution for a smaller and simpler system (or vice versa), and both systems have unique solutions per variable, then both systems specify the same process, one focusing on 'implementation details', and the other abstracting from these and focusing on the external behavior of the whole system. Comparing solutions of systems of recursive equations often plays a major role in process verification.

In Chapter 3 of this thesis we introduce an equivalence on recursively specified processes that is based on the *preservation* of solutions. This equivalence results from the theory of equivalences for regular systems of equations and applicative program schemes (cf. [31, 32]). Systems of (recursive) equations are considered with respect to their full sets of solutions in all models. As noted in [12], considering such a notion of equivalence avoids certain drawbacks of other methods used in process algebras, such as the restriction of process domains to ordered ones and considering only the least solution of a recursive system; or the restriction to systems that are guarded, and considering only the domains where all such systems have unique solutions (see [10]). In many cases, especially when data parameters are involved, such restrictions can be difficult to handle. For instance, it is not possible to justify transformations of recursive systems in value passing process algebras like μCRL using the method of restricting to syntactically guarded systems. For many models of processes (resembling different equivalences, see e.g. [44, 45]) guardedness becomes a more involved notion. Therefore it is useful to consider a model-independent equivalence of recursive systems in process algebra, and to use model specific equivalences only in cases where the former one is not sufficiently strong.

Typical for our approach is that we separate the question of unique solutions from the question how solutions of systems of recursive equations can be compared. This splitting of notions is worthwhile: properties of solutions are interesting for verification purposes, whereas comparison of systems of recursive equations is a fundamental notion that in itself can be applied in a model-independent way, only adhering to the axioms of process algebra.

1.5 Related Work

In context-free language theory, languages are considered as generated by a grammar. In fact a language is a set of strings in a certain finite terminal alphabet and a context-free grammar is a set of production rules transforming a non-terminal symbol into a string of terminal and non-terminal symbols. Starting from a singled-out non-terminal

A, the language generated by a grammar is the set of strings of terminal symbols that are generated from A. Grammar transformations are traditionally used in parsing and compiler construction. A context-free grammar is in *Greibach Normal Form (GNF)*, after Sheila Greibach [46], if its production rules are of the form $B \to a\alpha$, where B is a non-terminal symbol, a is a terminal symbol, and α is a string of zero or more non-terminal symbols. Every context-free language, not containing the empty string, can be generated by a grammar in GNF. The class of equations that have a GNF like representation is smaller if we consider them in the setting of process algebra. This class is restricted to *conditionally guarded* systems. For the precise definition of guardedness see Section 3.6.

In various process algebras, normal forms for closed terms were studied because of their use in proving completeness of the axiomatizations. These normal forms have a linear structure but without recursive calls, so that only finite terms can be constructed. For BPA and ACP, normal forms for closed terms can be found in [10], and the transformation of any closed term to normal form can be performed by term rewriting [6]. For timed μCRL, the transformation to normal forms can be found in [55]. In Sections 5.1.3 and 6.2.4 we use the functions like *simpl* which perform a kind of normal form transformation for terms.

Transformation procedures for systems of guarded Basic Process Algebra (BPA) equations to Greibach Normal Forms were outlined in [8] and presented in [72] and [62]. A transformation procedure for systems of guarded Basic Parallel Processes (BPP) equations to a similar kind of normal form was presented in [28, Appendix A]. To the best of our knowledge, a first description of a transformation of (non-parallel) pCRL into an LPE like format was given in [18]. In [23], a linearization procedure was sketched for a fragment of μCRL, which is similar to parallel pCRL, by means of an informal explanation and examples.

As has already been mentioned, many of the verification tools for concurrent systems, especially those based on imperative languages (for instance Spin [63]), make little use of program transformation techniques. They perform exhaustive exploration of all possible executions starting from the initial specification immediately. The CADP [38] toolset comes closest to the use of LPEs as an intermediate process representation. The CADP toolset is meant for analysis of specifications in LOTOS [66], which is a language based on a CCS-like process calculus and algebraic specifications of data types. In CADP a translation (cf. [42, 43, 41]) of a part of LOTOS similar to parallel pCRL to extended Petri Nets is used. Then, after some optimizations, an LTS is generated for further analysis. It is hard to compare this Petri-Nets based format with LPEs as the internal representation format of CADP is not publicly known.

The implications and equivalences of regular systems of recursive equations and recursive program schemes w.r.t. their full sets of solutions were studied extensively by Courcelle in [31, 32] and Benson and Guessarian in [12]. The definitions in these papers have a lot in common with our approach, but they could not be directly applied to the μCRL setting, due to the presence of binders, many-sortedness and other extensions to classical algebras used in μCRL.

1.6 Future Work

Most parts of the transformation procedures are proved to be correct. The correctness proofs of the final LPE definitions in Section 5.2 are sketched. The complexity of the data type manipulations is such that a mechanized proof checker is needed. As a consequence, the correctness of the final timed LPE definition in Section 6.3 has not been proved and, therefore, Theorem 6.3.1 is stated as a Conjecture.

The linearization algorithms presented in this thesis cover a large class of specifications in μCRL and timed μCRL. A class of conditionally guarded systems (Section 3.6) that can be transformed to a completely guarded form are not covered by the algorithms. For example, the n-parallel processes (cf. [60]) are conditionally guarded, but to make them syntactically guarded one needs to use properties of natural numbers. In general, it is undecidable whether a system of equations is conditionally guarded. Identifying decidable subsets of conditionally guarded systems could widen the "automatic" applicability of the linearization procedures.

A related problem is checking whether a certain equation in the system is reachable. Again, conditional reachability can be undecidable for certain data types, and identifying subsets of systems where reachability is decidable can help in reducing the sizes of LPEs obtained as a result of linearization.

Speaking about other ways to optimize the output of the linearization procedure, we describe the *regular linearization procedure*. By regular linearization we mean the linearization process that does not deploy infinite data types to encode process behavior. Regular linearization procedures can take the equations we have before the introduction of an infinite data type (the stack in Section 4.4 and LM in Section 5.2) and try to achieve the LPE form without this data type introduction. This is not always possible: for instance $\mathsf{X} = \mathsf{a}\cdot\mathsf{X}\cdot\mathsf{X}+\mathsf{a}$ cannot be linearized without introducing an infinite data type, even if we restrict to the bisimulation model. This follows from the fact that X represents an infinite graph in the bisimulation model (cf. [72]), but an LPE without infinite data types can only represent a finite graph in that model (cf. [39, page 40]). One of the possibilities for regular linearization is based on [72], and applies to the situation where regularity follows from the absence of termination in recursion, like in $\mathsf{X} = \mathsf{a}\cdot\mathsf{X}\cdot\mathsf{X}$. Restricting to standard process semantics for μCRL, an LPE that specifies the same behavior is $\mathsf{X} = \mathsf{a}\cdot\mathsf{X}$. However, this optimization is model dependent, as there are models in which the two equations have different sets of solutions. For some other cases, also dealt with in [72] and used in the μCRL Toolset, these optimizations can be justified on a general level using the equivalence of systems of process equations. For example, the system $G_1 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{Y}\cdot\mathsf{X}, \mathsf{Y} = \mathsf{b}\}$ can be transformed into $G_2 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{Z}, \mathsf{Z} = \mathsf{b}\cdot\mathsf{X}\}$, and we can prove that in every model the sets of solutions for X in both systems are equal, thus showing that this transformation is sound in every model. More on regular linearization, as it is implemented in the μCRL Toolset, can be found in [102, Section II.6].

Although μCRL incorporates most of the features that exist in other process algebras, some of the practically important ones are missing. These are, for instance, priorities [30] and interrupts [10, Section 6.1], process creation operators [17, Section 2.5], but one could also imagine probabilistic [4, 67], stochastic [61] and hybrid [34] extensions to μCRL. Mixing data and processes in a way that is done in the π-

calculus [81] could lead to a higher-order μCRL. The linearization procedure could, quite likely, be extended to handle these features in a way that is done for the timed extension of μCRL in Chapter 6.

For many of the above mentioned features, as well as for some of the already present ones in μCRL, a further development of the universal-algebraic theory of abstract data types is required. The process algebras BPA and ACP require just single-sorted equational logic to be axiomatically dealt with. A rather straightforward extension of ACP to μCRL requires a more sophisticated universal-algebraic underpinning. Below we present some of the possible extensions.

- *Conditional equational logic and equational Horn logic (cf. [75, Section 5.3]).* These logics allow to express finite implications and negations of equational identities. In standard definitions of μCRL and timed μCRL and in the linearization procedures presented in this thesis, the axioms that require these extensions are not used, but a user may feel the need to specify such a data type.

- *Full many-sorted first-order logic with equality.* It covers all of the extensions of equational logic, including induction principles and binders. Unfortunately there are not so many fully automatic provers for this logic; human interaction is often needed.

- *Partial (cf. [1]) and non-deterministic data types.* In many cases these data types are used informally (for instance $pred(0)$ is not interpreted in the standard model of the natural numbers). For a treatment of such data types we refer to [100].

From a practical point of view, development of term rewriting [84] and theorem proving [83] techniques are important to efficiently handle the kind of data types that are used in μCRL specifications and added during linearization. To this end, a library of efficient basic data types for μCRL is important (cf. [59]). For the implementation of the linearization procedure some practical considerations, such as complexity analysis, relevance of optimizations like rewriting of data terms and reachability analysis after each step of the procedure, clustering of actions, etc., are still needed. An implementation of the linearization procedure of Chapter 5 using rewriting strategies [98] is currently under development (cf. [91]).

1.7 Structure of the Thesis

The remainder of this thesis has the following structure. In Chapter 2 we present the definition of μCRL as an algebraic theory (for the language definition we refer to [53, 47]). In Chapter 3, which is an extended version of [86], systems of recursive equations in μCRL and a notion of equivalence on them is presented. Chapter 4, which is based on [54], contains the linearization procedure for parallel pCRL. In Chapter 5, based on [96, 95], the linearization procedure is extended to the full μCRL setting. Finally, Chapter 6, based on [89], deals with an extension of the linearization procedure to the setting of timed μCRL.

Appendix A contains detailed descriptions of the resulting LPEs that involve renaming operations of μCRL. Appendix B contains the axioms of timed μCRL and proofs of identities valid in timed μCRL that are needed to show the correctness of the linearization steps. Appendix C contains the full source code listing of the data type definitions used in Sections 5.2 and 6.3.

Chapter 2

Algebraic Theory of μCRL

The language μCRL was introduced in [53] as a combination of "static" data types defined by means of algebraic specifications (cf. [14, 68]), and processes defined by means of recursive equations in process algebra (cf. [10, 39]). In that paper the syntax and static semantics of the language is introduced and a decidable class of well-formed μCRL specifications is identified. Furthermore, an algebraic semantics of well-formed specifications in μCRL is presented there by identifying a class of algebras that serve as models for data definitions, and constructing a transition system specification for processes using Structured Operational Semantics (SOS) rules (cf. [2]).

In [52] a proof theory based on an extension of the equational calculus with induction and recursion principles for the models of μCRL is presented. The definition of summation over data domains (or choice quantification) in that paper was based on an axiomatization in first-order logic with equality. Bas Luttik [69] used generalized algebras of Rasiova and Sikorski [88] to axiomatize choice quantification. However, these algebras have no syntactic characterization, and the generalized equational calculus presented in [69] was proven sound only for pCRL. In [69] there is also a cylindric-algebraic axiomatization of choice quantification, but because it introduces a completely different notation and is worked out only for pCRL, we do not use it here. In [76] the binding algebras of Yong Sun [94] were used to investigate SOS rules for binders and to define a summation operator for real-time process algebras which is similar to choice quantification. In this thesis we use the latter approach for the formal treatment of choice quantification and parameterized process equations.

In this chapter we present the algebraic theory of μCRL by defining the sorts of booleans and processes, by presenting their signatures and axioms. Furthermore we derive useful identities for these sorts that are used in the later chapters. In Chapter 6 we extend this theory to timed μCRL by defining the sort of time and adapting the sort of processes. In Chapter 3 we consider systems of equations and parameterized equations for processes and investigate the use of the above mentioned calculi to syntactically characterize preservation of solutions of these systems of equations.

2.1 Syntax and Axioms of μCRL

As already mentioned, μCRL specifications contain algebraic specifications of several abstract data types. The only data type that is required are booleans. Therefore we start by presenting a specification of booleans as a single sorted algebraic specification.

Booleans. First we define the signature of booleans and axioms for booleans, which are quite standard and can be found for instance in [26, Chapter IV].

Definition 2.1.1. The signature of *Bool* consists of constants $\mathbf{t}$ and $\mathbf{f}$, unary operation *not*, binary operations *and*, *or*, *eq* and *impl*, and a ternary operation *if*.

Note (Booleans). We use infix notation $\neg$, $\wedge$, $\vee$, $\leftrightarrow$, $\rightarrow$ for *not*, *and*, *or*, *eq*, *impl*, respectively.

Note (Axioms). Throughout this thesis we use the symbol $\approx$ to represent the main predicate symbol of the (binding) equational calculus. The usual equality symbol ($=$) we use to define recursive equations (cf. Chapter 3), as well as to abbreviate the built-in equality function symbols *eq*.

Definition 2.1.2. The axioms of *Bool* are the identities presented in Table 2.1.

$x \wedge y \approx y \wedge x$	$x \vee y \approx y \vee x$
$(x \wedge y) \wedge z \approx x \wedge (y \wedge z)$	$(x \vee y) \vee z \approx x \vee (y \vee z)$
$x \wedge x \approx x$	$x \vee x \approx x$
$x \wedge (x \vee y) \approx x$	$x \vee (x \wedge y) \approx x$
$(x \wedge y) \vee (x \wedge z) \approx x \wedge (y \vee z)$	$(x \vee y) \wedge (x \vee z) \approx x \vee (y \wedge z)$
$x \wedge \mathbf{f} \approx \mathbf{f}$	$x \vee \mathbf{t} \approx \mathbf{t}$
$x \wedge \neg x \approx \mathbf{f}$	$x \vee \neg x \approx \mathbf{t}$
$x \leftrightarrow y \approx (x \wedge y) \vee (\neg x \wedge \neg y)$	$x \rightarrow y \approx \neg x \vee y$
$\mathit{if}(x, y, z) \approx (x \wedge y) \vee (\neg x \wedge z)$	

Table 2.1: Axioms of *Bool*.

Together with the inference rules of equational calculus, the axioms in Table 2.1 form the *calculus of boolean identities*. The identities of the following lemma will be used extensively throughout this thesis.

Lemma 2.1.3. *The following identities are derivable in the calculus of boolean identities.*

- $\neg\neg x \approx x$;
- $\neg(x \wedge y) \approx \neg x \vee \neg y$;
- $\neg(x \vee y) \approx \neg x \wedge \neg y$;
- $x \wedge (\neg x \vee y) \approx x \wedge y$;

- $x \vee (\neg x \wedge y) \approx x \vee y$;
- $x \leftrightarrow x \approx \mathbf{t}$;
- $x \rightarrow x \approx \mathbf{t}$;
- $x \leftrightarrow y \approx y \leftrightarrow x$;
- $x \rightarrow y \wedge y \rightarrow x \approx x \leftrightarrow y$
- $x \leftrightarrow y \wedge y \leftrightarrow z \approx x \leftrightarrow y \wedge y \leftrightarrow z \wedge x \leftrightarrow z$;
- $x \rightarrow y \wedge y \rightarrow z \approx x \rightarrow y \wedge y \rightarrow z \wedge x \rightarrow z$;
- $\mathit{if}(x \leftrightarrow y, x, y) \approx y$;
- *if* $x \leftrightarrow y \approx \mathbf{t}$, *then* $x \approx y$.
- *if* $x \approx y$, *then* $x \leftrightarrow y \approx \mathbf{t}$.

Proof. The first three identities are proved in [26, Chapter IV]. All the rest are simple exercises. We prove the last three:

- $\mathit{if}(x \leftrightarrow y, x, y) \approx y$;

$$\begin{aligned}
\mathit{if}(x \leftrightarrow y, x, y) &\approx ((x \leftrightarrow y) \wedge x) \vee (\neg(x \leftrightarrow y) \wedge y) \\
&\approx (((x \wedge y) \vee (\neg x \wedge \neg y)) \wedge x) \vee (\neg((x \wedge y) \vee (\neg x \wedge \neg y)) \wedge y) \\
&\approx ((x \wedge y \wedge x) \vee (\neg x \wedge \neg y \wedge x)) \vee ((\neg x \vee \neg y) \wedge (x \vee y) \wedge y) \\
&\approx ((x \wedge y) \vee \mathbf{f}) \vee ((\neg x \vee \neg y) \wedge y) \\
&\approx (x \wedge y) \vee (\neg x \wedge y) \\
&\approx (x \vee \neg x) \wedge y \\
&\approx \mathbf{t} \wedge y \\
&\approx y
\end{aligned}$$

- if $x \leftrightarrow y \approx \mathbf{t}$, then $x \approx y$.

$$\begin{aligned}
y &\approx \mathit{if}(x \leftrightarrow y, x, y) \\
&\approx \mathit{if}(\mathbf{t}, x, y) \\
&\approx (\mathbf{t} \wedge x) \vee (\mathbf{f} \wedge y) \\
&\approx x
\end{aligned}$$

  We note that this implication is derivable for any data type where $\mathit{if}(\mathit{eq}(x, y), x, y) \approx y$ and $\mathit{if}(\mathbf{t}, x, y) \approx x$ are derivable.

- if $x \approx y$, then $x \leftrightarrow y \approx \mathbf{t}$.

$$\begin{aligned}
x \leftrightarrow y &\approx x \leftrightarrow x \\
&\approx \mathbf{t}
\end{aligned}$$

  We note that this implication is derivable for any data type where $\mathit{eq}(x, x) \approx \mathbf{t}$ is derivable.

□

It is well-known that the class of algebras for the boolean signature in which all the boolean identities hold forms a *variety* (cf. [26, Chapter II]). According to [53] only the algebras where $\mathbf{t}$ and $\mathbf{f}$ represent two different elements are interesting models for booleans in μCRL. It is easy to see that such boolean algebras exist.

Other Data Types. Any other data type in μCRL is specified in a similar way by providing a signature and axioms from which all other identities are derived. Other data sorts have generally different axioms, and sometimes induction principles (cf. [59]) are required to describe them. Properties of some domains cannot be fully described using (a kind of) equational calculus (less expressive than first-order logic with equality). In a nutshell, a domain D is inductive if equality of two open terms t and u ($D \models t \approx u$) holds whenever for all closed term substitutions σ the equality of $\sigma(t)$ and $\sigma(u)$ holds. Such a domain is said to have no "junk", because all of its elements can be addressed by interpreting closed terms. In some cases the set of all closed terms is too big and we select a sub-signature of constructors that generate all distinct closed terms. In fact, any set of closed terms may be selected to represent the terms in *normal form*, as long as all other closed terms are equal to a normal form. The induction principles of structured, constructor and normal-form induction are syntactic derivation rules that extend equational calculi, to stand closer to first-order logic with equality in terms of derivative power.

Processes. The sort of processes is specified by describing the set of identities in a similar way as it is done for the data types. One noticeable difference is the presence of binders $\sum_{d:D}$, which requires modifications to the inference rules of equational calculus, as well as to the choice of semantic domain, being binding algebras of [94]. More importantly, recursion (cf. Chapter 3), which is used to define processes in μCRL, takes us out of initial algebra semantics. We define the binding-equational calculus of μCRL by defining its signature and axioms, and we use the inference rules of the calculus $\vdash_{eBA}$ presented in [94, Section 1]. Many of the axioms are taken from, or inspired by, [49, 52].

Note (Vector Notation). Tuples occur a lot in the language, so we use a vector notation for them. An expression $\overrightarrow{d}$ is an abbreviation for $d^1, \ldots, d^n$, where the d^k are data variables. Similarly, if type information is given, $\overrightarrow{d{:}D}$ is an abbreviation for $d^1{:}D^1, \ldots, d^n{:}D^n$ for some natural number n. In case $n = 0$, the whole vector vanishes as well as brackets (if any) surrounding it. For instance, $\mathsf{a}(\overrightarrow{d})$ is an abbreviation for a in this case (here a is an *action*, this notion is introduced below). For all vectors $\overrightarrow{d}$ and $\overrightarrow{e}$ we have $\overrightarrow{d}, \overrightarrow{e} = \overrightarrow{d, e}$. Thus $\overrightarrow{d, e}$ is an abbreviation for $d^1, \ldots, d^n, e^1, \ldots, e^{n'}$. We also write $\overrightarrow{d{:}D}$ & $e{:}E$ for $d^1{:}D^1, \ldots, d^n{:}D^n, e{:}E$.

For any vector of variables $\overrightarrow{d}$, $\overrightarrow{f}(\overrightarrow{d})$ is an abbreviation for $f^1(\overrightarrow{d}), \ldots, f^m(\overrightarrow{d})$ for some $m \in \mathit{Nat}$ and $\overrightarrow{f} = f^1, \ldots, f^m$, where each $f^k(\overrightarrow{d})$ is a data term that may contain elements of $\overrightarrow{d}$ as free variables. As with vectors of variables, in case $m = 0$ the vector of data terms vanishes. We often use $\overrightarrow{t}$ to express a data term vector without explicitly denoting its variables.

Definition 2.1.4 (Signature of μCRL). The signature of μCRL consists of data sorts (or 'data types') including *Bool* as defined above, and a distinct sort *Proc* of processes. Each data sort D is assumed to be equipped with a binary function $eq : D \times D \to \mathit{Bool}$. (This requirement can be weakened by demanding such functions only for data sorts that are parameters of communicating actions.) The operational signature of μCRL is parameterized by the finite set of action labels *ActLab* and a partial commutative and associative function $\gamma : ActLab \times ActLab \rightarrow ActLab$ such that $\gamma(\mathsf{a}_1, \mathsf{a}_2) \in ActLab$ implies that $\mathsf{a}_1, \mathsf{a}_2$ and $\gamma(\mathsf{a}_1, \mathsf{a}_2)$ have parameters of the same sorts. The process operations are the ones listed below:

- actions $\mathsf{a}(\overrightarrow{t})$ parameterized by data terms $\overrightarrow{t}$, where $\mathsf{a} \in ActLab$ is an action label. More precisely, a is an operation $\mathsf{a} : \overrightarrow{D_\mathsf{a}} \rightarrow Proc$. We write $type(\mathsf{a})$ for $\overrightarrow{D_\mathsf{a}}$.
- constants δ and τ of sort *Proc*.
- binary operations $+, \cdot, \|, \mathbin{\|\!\_}, |$ defined on *Proc*, where $|$ is defined using γ.
- unary *Proc* operations $\partial_H, \tau_I, \rho_R$ for each set of action labels $H, I \subseteq ActLab$ and an action label renaming function $R : ActLab \rightarrow ActLab$ such that a and $R(\mathsf{a})$ have parameters of the same sorts. Such functions R we call *well-defined* action label renaming functions.
- a ternary operation $\_ \lhd \_ \rhd \_ : Proc \times Bool \times Proc \rightarrow Proc$.
- binders $\sum_{d:D} : D.Proc \rightarrow Proc$, for each data sort D.

The partial commutative and associative function γ is called a *communication function.* If $\gamma(\mathsf{a}, \mathsf{b}) = \mathsf{c}$, this indicates that actions with labels a and b can synchronize, becoming action c, provided that the data parameters of these actions are equal. The case when $\gamma(\gamma(\mathsf{a}, \mathsf{b}), \mathsf{c})$ is undefined for all a,b and c, which means that at most two parties can communicate synchronously, is called *handshaking* communication (or simply handshaking). The constant δ represents a deadlocked process and the constant τ represents some internal or hidden activity. The *choice* operator $+$ and the *sequential composition* operator $\cdot$ are well known. The *merge* operator $\|$ represents *parallel composition.* The $\mathbin{\|\!\_}$ (*left merge*) and $|$ (*communication merge*) are auxiliary operations used to equationally define $\|$. The *encapsulation* operator $\partial_H(q)$ blocks actions in q with action labels in the set H, which is especially used to enforce actions to communicate. The *hiding* operator $\tau_I(q)$ with a set of action labels $I = \{\mathsf{a}, \mathsf{b}, \dots\}$ hides actions with these labels in q by renaming them to τ. The *renaming* operator $\rho_R(q)$ where R is a function from action labels to action labels renames each action with label a in q to an action with label $R(\mathsf{a})$. The operator $p_1 \lhd c \rhd p_2$ is the *if* c *then* p_1 *else* p_2 operator, where c is an expression of type *Bool*. The *sum operator* $\sum_{d:D} p$ expresses a (potentially infinite) choice $p[d := d_0] + p[d := d_1] + \cdots$ if data domain $D = \{d_0, d_1, \dots\}$, and $p[d := d_i]$ is the term p with all free occurrences of d replaced by d_i.

Definition 2.1.5 (Axioms of μCRL). Axioms of μCRL are the ones presented in Tables 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 and 2.8. We assume that

- the descending order of binding strength of operators is: $\cdot$, $\{\|, \mathbin{\|\!\_}, |\}$, $\lhd\rhd$, $\sum$, $+$;
- x, y, z are variables of sort *Proc*;
- c, c_1, c_2 are variables of sort *Bool*;
- $d, d^1, d^n, d', \dots$ are data variables (but d in $\sum_{d:D}$ is not a variable);

- b stands for either $\mathsf{a}(\overrightarrow{d})$, or τ, or δ;
- $\overrightarrow{d} = \overrightarrow{d'}$ is an abbreviation for $eq(d^1, d'^1) \wedge \cdots \wedge eq(d^n, d'^n)$, when $\overrightarrow{d} = d^1, \ldots, d^n$ and $\overrightarrow{d'} = d'^1, \ldots, d'^n$;
- the axioms where p and q occur are schemata ranging over all terms p and q of sort *Proc*, *including* those in which d occurs freely;
- the axiom (SUM2) is a scheme ranging over all terms r of sort *Proc* in which d *does not* occur freely.

$x + y \approx y + x$	(A1)
$x + (y + z) \approx (x + y) + z$	(A2)
$x + x \approx x$	(A3)
$(x + y) \cdot z \approx x \cdot z + y \cdot z$	(A4)
$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$	(A5)
$x + \delta \approx x$	(A6)
$\delta \cdot x \approx \delta$	(A7)
$x \cdot \tau \approx x$	(B1)
$z \cdot (\tau \cdot (x + y) + x) \approx z \cdot (x + y)$	(B2)

Table 2.2: Basic axioms of μCRL.

$x \parallel y \approx (x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x) + x \mid y$	(CM1)
$b \mathbin{\lfloor\!\lfloor} x \approx b \cdot x$	(CM2)
$(b \cdot x) \mathbin{\lfloor\!\lfloor} y \approx b \cdot (x \parallel y)$	(CM3)
$(x + y) \mathbin{\lfloor\!\lfloor} z \approx x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	(CM4)
$(b \cdot x) \mid b' \approx (b \mid b') \cdot x$	(CM5)
$(b \cdot x) \mid (b' \cdot y) \approx (b \mid b') \cdot (x \parallel y)$	(CM7)
$(x + y) \mid z \approx x \mid z + y \mid z$	(CM8)
$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) \approx \gamma(\mathsf{a}, \mathsf{a}')(\overrightarrow{d}) \lhd \overrightarrow{d} = \overrightarrow{d'} \rhd \delta$ if $\gamma(\mathsf{a}, \mathsf{a}')$ is defined	(CF1)
$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) \approx \delta$ otherwise	(CF2)
$\tau \mid b \approx \delta$	(CT1)
$x \mid y \approx y \mid x$	(SC3)

Table 2.3: Axioms for parallel composition in μCRL.

The axioms (B1) and (B2) are not used in the transformations described in this thesis, so these transformations are also sound in models where these two axioms do not hold.

To prove identities in μCRL we use a combined many-sorted calculus, which for the sort of processes has the rules of binding-equational calculus (cf. [94, Section 1]),

$x \lhd \mathbf{t} \rhd y \approx x$	(Cond1)
$x \lhd \mathbf{f} \rhd y \approx y$	(Cond2)
$x \lhd c \rhd y \approx x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta$	(Cond3)
$(x \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta \approx (x \lhd c_1 \wedge c_2 \rhd \delta)$	(Cond4)
$(x \lhd c_1 \rhd \delta) + (x \lhd c_2 \rhd \delta) \approx x \lhd c_1 \vee c_2 \rhd \delta$	(Cond5)
$(x \lhd c \rhd \delta) \cdot y \approx (x \cdot y) \lhd c \rhd \delta$	(Cond6)
$(x + y) \lhd c \rhd \delta \approx x \lhd c \rhd \delta + y \lhd c \rhd \delta$	(Cond7)
$(x \lhd c \rhd \delta) \mathbin{\|\!\_} y \approx (x \mathbin{\|\!\_} y) \lhd c \rhd \delta$	(Cond8)
$(x \lhd c \rhd \delta) \mid y \approx (x \mid y) \lhd c \rhd \delta$	(Cond9)
$(x \lhd c \rhd \delta) \cdot (y \lhd c \rhd \delta) \approx (x \cdot y) \lhd c \rhd \delta$	(Sca)
$p \lhd eq(d, e) \rhd \delta \approx p[e := d] \lhd eq(d, e) \rhd \delta$	(PE)

Table 2.4: Axioms for conditions in μCRL.

$\sum_{d:D} x \approx x$	(SUM1)
$\sum_{e:D} r \approx \sum_{d:D} (r[e := d])$	(SUM2)
$\sum_{d:D} p \approx \sum_{d:D} p + p$	(SUM3)
$\sum_{d:D} (p + q) \approx \sum_{d:D} p + \sum_{d:D} q$	(SUM4)
$\sum_{d:D} (p \cdot x) \approx (\sum_{d:D} p) \cdot x$	(SUM5)
$\sum_{d:D} (p \mathbin{\|\!\_} x) \approx (\sum_{d:D} p) \mathbin{\|\!\_} x$	(SUM6)
$\sum_{d:D} (p \mid x) \approx (\sum_{d:D} p) \mid x$	(SUM7)
$\sum_{d:D} (\partial_H(p)) \approx \partial_H(\sum_{d:D} p)$	(SUM8)
$\sum_{d:D} (\tau_I(p)) \approx \tau_I(\sum_{d:D} p)$	(SUM9)
$\sum_{d:D} (\rho_R(p)) \approx \rho_R(\sum_{d:D} p)$	(SUM10)
$\sum_{d:D} (p \lhd c \rhd \delta) \approx (\sum_{d:D} p) \lhd c \rhd \delta$	(SUM12)

Table 2.5: Axioms for sums in μCRL.

for the sort of booleans has the rules of equational calculus, while other data sorts may include induction principles (cf. [99, Chapters 5 and 6]) which could be used to derive process identities as well. We note that the derivation rules of binding-equational calculus do not allow to substitute terms containing free variables if they become bound. For example, in axiom (SUM1) we cannot substitute $\mathsf{a}(d)$ for x.

$\partial_H(b) \approx b$ if $b = \tau$ or ($b = \mathsf{a}(\vec{d})$ and $\mathsf{a} \notin H$)	(D1)
$\partial_H(b) \approx \delta$ otherwise	(D2)
$\partial_H(x+y) \approx \partial_H(x) + \partial_H(y)$	(D3)
$\partial_H(x \cdot y) \approx \partial_H(x) \cdot \partial_H(y)$	(D4)
$\partial_H(x \lhd c \rhd \delta) \approx \partial_H(x) \lhd c \rhd \delta$	(D5)
$\tau_I(b) \approx b$ if $b = \delta$ or ($b = \mathsf{a}(\vec{d})$ and $\mathsf{a} \notin I$)	(T1)
$\tau_I(b) \approx \tau$ otherwise	(T2)
$\tau_I(x+y) \approx \tau_I(x) + \tau_I(y)$	(T3)
$\tau_I(x \cdot y) \approx \tau_I(x) \cdot \tau_I(y)$	(T4)
$\tau_I(x \lhd c \rhd \delta) \approx \tau_I(x) \lhd c \rhd \delta$	(T5)
$\rho_R(\delta) \approx \delta$	(RD)
$\rho_R(\tau) \approx \tau$	(RT)
$\rho_R(\mathsf{a}(\vec{d})) \approx R(\mathsf{a})(\vec{d})$	(R1)
$\rho_R(x+y) \approx \rho_R(x) + \rho_R(y)$	(R3)
$\rho_R(x \cdot y) \approx \rho_R(x) \cdot \rho_R(y)$	(R4)
$\rho_R(x \lhd c \rhd \delta) \approx \rho_R(x) \lhd c \rhd \delta$	(R5)

Table 2.6: Axioms for renaming operators in μCRL.

$(x \mathbin{\|\!\lfloor} y) \mathbin{\|\!\lfloor} z \approx x \mathbin{\|\!\lfloor} (y \parallel z)$	(SC1)
$(x \mid y) \mid z \approx x \mid (y \mid z)$	(SC4)
$x \mid (y \mathbin{\|\!\lfloor} z) \approx (x \mid y) \mathbin{\|\!\lfloor} z$	(SC5)
$x \mathbin{\|\!\lfloor} \delta \approx x \cdot \delta$	(SCD1)
$x \mid \delta \approx \delta$	(SCD2)

Table 2.7: Axioms for Standard Concurrency in μCRL.

Definition 2.1.6. Two process terms p_1 and p_2 are *(unconditionally) equivalent* (notation $p_1 \approx p_2$) if $p_1 \approx p_2$ is derivable from the axioms of μCRL and boolean identities by using many sorted binding-equational calculus. In this case we write $\{\mu\text{CRL}, \mathit{BOOL}\} \vdash_{eBA} p_1 \approx p_2$. Here *BOOL* is used to refer to the specification of the booleans, and the use of equational logic for deriving boolean identities.

Two process terms p_1 and p_2 are *conditionally equivalent* if

$$\{\mu\text{CRL}, \mathit{BOOL}, \mathit{DATA}\} \vdash_{eBA} p_1 \approx p_2.$$

Here *DATA* is used to refer to the specification of all data sorts involved, and all proof rules that may be applied.

2.2 Derivable Identities in μCRL

A number of identities that can be found as axioms of μCRL, for instance in [52], are derivable in our setting, but, nevertheless, we shall still call them axioms of μCRL.

$\partial_{H_1}(\partial_{H_2}(x)) \approx \partial_{H_1 \cup H_2}(x)$	(DD)
$\tau_{I_1}(\tau_{I_2}(x)) \approx \tau_{I_1 \cup I_2}(x)$	(TT)
$\rho_{R_1}(\rho_{R_2}(x)) \approx \rho_{R_1 \circ R_2}(x)$	(RR)
$\partial_H(\tau_I(x)) \approx \tau_I(\partial_{H \setminus I}(x))$	(DT)
$\partial_H(\rho_R(x)) \approx \rho_R(\partial_{R^{-1}(H)}(x))$	(DR)
$\tau_I(\rho_R(x)) \approx \rho_R(\tau_{R^{-1}(I)}(x))$	(TR)
$\partial_\emptyset(x) \approx x$	(D0)
$\tau_\emptyset(x) \approx x$	(T0)
$\rho_{R_{ActLab}}(x) \approx x$	(R0)
$\tau_I(\partial_H(x)) \approx \tau_{I \setminus H}(\partial_H(x))$	(TDO)
$\rho_R(\tau_I(x)) \approx \rho_{R_I}(\tau_I(x))$	(RTO)

where $R_S(\mathsf{a})$ for $S \subseteq ActLab$ is defined to be equal to a if $\mathsf{a} \in S$ and to $R(\mathsf{a})$ otherwise.

Table 2.8: Axioms for combinations of renaming operators.

Lemma 2.2.1. *The following identities are derivable from the axioms of μCRL.*

$$b \mid (b' \cdot x) \approx (b \mid b') \cdot x \tag{CM6}$$
$$x \mid (y + z) \approx x \mid y + x \mid z \tag{CM9}$$
$$b \mid \tau \approx \delta \tag{CT2}$$
$$\delta \mid b \approx \delta \tag{CD1}$$
$$b \mid \delta \approx \delta \tag{CD2}$$
$$x \mid (y \lhd c \rhd \delta) \approx (x \mid y) \lhd c \rhd \delta \tag{Cond9$'$}$$
$$\sum_{d:D} (x \mid p) \approx x \mid (\sum_{d:D} p) \tag{SUM7$'$}$$

Proof. The axiom (CD2) is a special instance of (SCD2), and the rest are derivable from the symmetric axioms using (SC3). □

A process p is a *subprocess* of q if $q \approx p + q$ (cf. [10]). If $q \approx p + r$ for some process r, then

$$p + q \approx p + p + r \overset{(A3)}{\approx} p + r \approx q$$

which means that p is a subprocess of q. The following lemma says that if two processes are subprocesses of each other, then they are equal. This gives us a useful proof method for processes.

Lemma 2.2.2. *If for some process terms p, q, r, s $p \approx q + r$ and $q \approx p + s$, then $p \approx q$.*

Proof.

$$\begin{aligned} p + q &\approx p + p + s \\ \text{(A3)} &\approx p + s \\ &\approx q \end{aligned} \qquad \text{and} \qquad \begin{aligned} p + q &\approx q + r + q \\ \text{(A1)} &\approx q + q + r \\ \text{(A3)} &\approx q + r \\ &\approx p \end{aligned}$$

□

The following lemma is similar to Lemma 5.44 of [69, pages 98–99]. It says that if the summation variable occurs only in the condition, then in some cases the sum can be eliminated.

Lemma 2.2.3. *If for some term dd_0 of sort D that does not have free occurrences of d, and for some boolean term cc we have $cc \wedge cc[d := dd_0] \approx cc$, then*

$$\sum_{d:D} x \lhd cc \rhd \delta \approx x \lhd cc[d := dd_0] \rhd \delta$$

Proof.

$$\begin{aligned}
&\sum_{d:D} x \lhd cc \rhd \delta \\
&\qquad (\text{SUM3}) \approx \sum_{d:D} x \lhd cc \rhd \delta + x \lhd cc[d := dd_0] \rhd \delta \\
&\qquad (\text{A1}) \approx x \lhd cc[d := dd_0] \rhd \delta + \sum_{d:D} x \lhd cc \rhd \delta
\end{aligned}$$

and

$$\begin{aligned}
&x \lhd cc[d := dd_0] \rhd \delta \\
&\qquad (\text{SUM1}) \approx \sum_{d:D} x \lhd cc[d := dd_0] \rhd \delta \\
&\qquad \approx \sum_{d:D} x \lhd cc[d := dd_0] \wedge \mathbf{t} \rhd \delta \\
&\qquad \approx \sum_{d:D} x \lhd cc[d := dd_0] \wedge (cc \vee \neg cc) \rhd \delta \\
&\qquad \approx \sum_{d:D} x \lhd (cc[d := dd_0] \wedge cc) \vee (cc[d := dd_0] \wedge \neg cc) \rhd \delta \\
&\qquad (\text{Cond5}), (\text{SUM4}) \approx \sum_{d:D} x \lhd cc[d := dd_0] \wedge cc \rhd \delta + \sum_{d:D} x \lhd cc[d := dd_0] \wedge \neg cc \rhd \delta \\
&\qquad (\text{Assumption}) \approx \sum_{d:D} x \lhd cc \rhd \delta + \sum_{d:D} x \lhd cc[d := dd_0] \wedge \neg cc \rhd \delta
\end{aligned}$$

By Lemma 2.2.2 we get the desired identity. □

Lemma 2.2.4 (Sum Elimination). *The following identities are derived with the help of the previous lemma.*

1. $\sum_{d:D} x \lhd eq(d,e) \rhd \delta \approx x$;
2. $\sum_{d:D} p \lhd eq(d,e) \rhd \delta \approx p[d := e]$;

Proof. The first identity is a direct applications of Lemma 2.2.3 (by taking $dd_0 = e$). The second identity follows from (PE) and the first identity. □

The following lemma is similar to Proposition 3.1.ii of [49]. It says that the order of sums is not important.

Lemma 2.2.5 (Sum Commutativity). $\sum_{d:D}\sum_{e:E} p \approx \sum_{e:E}\sum_{d:D} p$

Proof.

$$\begin{aligned}
&\sum_{d:D}\sum_{e:E} p\\
&(\mathrm{SUM1}) \approx \sum_{e:E}\sum_{d:D}(\sum_{d:D}\sum_{e:E} p)\\
&(\mathrm{SUM3}) \approx \sum_{e:E}\sum_{d:D}(\sum_{d:D}\sum_{e:E} p + p)\\
&(\mathrm{SUM4}) \approx \sum_{e:E}\sum_{d:D}(\sum_{d:D}\sum_{e:E} p) + \sum_{e:E}\sum_{d:D} p\\
&(\mathrm{SUM1}) \approx \sum_{d:D}\sum_{e:E} p + \sum_{e:E}\sum_{d:D} p
\end{aligned}$$

In a similar way we obtain

$$\sum_{e:E}\sum_{d:D} p \approx \sum_{e:E}\sum_{d:D} p + \sum_{d:D}\sum_{e:E} p.$$

By Lemma 2.2.2 we get the desired identity. □

The identities of the following lemma are used in several steps of the linearization procedures throughout this thesis (for instance for simple term rewriting in Sections 4.2.2, 5.1.1, and 6.2.2).

Lemma 2.2.6 (Derivable Identities in μCRL). *The following identities are derivable from the axioms of μCRL and booleans:*

1. $x \lhd c \rhd x \approx x$;
2. $x \lhd c \rhd y \approx y \lhd \neg c \rhd x$;
3. $(x \lhd c \rhd y) \lhd c \rhd \delta \approx x \lhd c \rhd \delta$;
4. $(x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \approx x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2$;
5. $x \cdot (y \lhd c \rhd z) \approx x \cdot y \lhd c \rhd x \cdot z$;
6. $x \parallel y \approx y \parallel x$;
7. $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$;
8. $x \parallel \delta \approx x \cdot \delta$;
9. $x \parallel (y \cdot \delta) \approx (x \parallel y) \cdot \delta$;
10. $\tau_I(\partial_H(x)) \approx \tau_{I\cup H}(\partial_H(x))$;

11. $\rho_R(\tau_I(\partial_H(x))) \approx \rho_{R_{I\cup H}}(\tau_I(\partial_H(x)))$;

12. $\rho_R(\partial_H(x)) \approx \rho_{R_H}(\partial_H(x))$.

Proof. 1. $x \lhd c \rhd x \approx x$;

$$\begin{aligned}
& x \lhd c \rhd x \\
(\text{Cond3}) &\approx x \lhd c \rhd \delta + x \lhd \neg c \rhd \delta \\
(\text{Cond5}) &\approx x \lhd c \vee \neg c \rhd \delta \\
(\text{Cond1}) &\approx x
\end{aligned}$$

2. $x \lhd c \rhd y \approx y \lhd \neg c \rhd x$;

$$\begin{aligned}
& x \lhd c \rhd y \\
(\text{Cond3}) &\approx x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta \\
(\text{A1}) &\approx y \lhd \neg c \rhd \delta + x \lhd \neg\neg c \rhd \delta \\
(\text{Cond3}) &\approx y \lhd \neg c \rhd x
\end{aligned}$$

3. $(x \lhd c \rhd y) \lhd c \rhd \delta \approx x \lhd c \rhd \delta$;

$$\begin{aligned}
& (x \lhd c \rhd y) \lhd c \rhd \delta \\
(\text{Cond3}) &\approx (x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta) \lhd c \rhd \delta \\
(\text{Cond7}), (\text{Cond4}) &\approx x \lhd c \wedge c \rhd \delta + y \lhd \neg c \wedge c \rhd \delta \\
&\approx x \lhd c \rhd \delta + y \lhd \mathbf{f} \rhd \delta \\
(\text{Cond2}), (\text{A6}) &\approx x \lhd c \rhd \delta
\end{aligned}$$

4. $(x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \approx x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2$;

$$\begin{aligned}
& (x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \\
(\text{Cond3}) &\approx (x_1 \lhd c \rhd \delta + x_2 \lhd \neg c \rhd \delta) \cdot (y_1 \lhd c \rhd y_2) \\
(\text{A4}), (\text{Cond6}), (2) &\approx x_1 \cdot (y_1 \lhd c \rhd y_2) \lhd c \rhd \delta + x_2 \cdot (y_2 \lhd \neg c \rhd y_1) \lhd \neg c \rhd \delta \\
(\text{Sca}) &\approx (x_1 \lhd c \rhd \delta) \cdot ((y_1 \lhd c \rhd y_2) \lhd c \rhd \delta) + \\
& \quad (x_2 \lhd \neg c \rhd \delta) \cdot ((y_2 \lhd \neg c \rhd y_1) \lhd \neg c \rhd \delta) \\
(3) &\approx (x_1 \lhd c \rhd \delta) \cdot (y_1 \lhd c \rhd \delta) + (x_2 \lhd \neg c \rhd \delta) \cdot (y_2 \lhd \neg c \rhd \delta) \\
(\text{Sca}) &\approx x_1 \cdot y_1 \lhd c \rhd \delta + x_2 \cdot y_2 \lhd \neg c \rhd \delta \\
(\text{Cond3}) &\approx x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2
\end{aligned}$$

5. $x \cdot (y \lhd c \rhd z) \approx x \cdot y \lhd c \rhd x \cdot z$;

$$\begin{aligned}
& x \cdot (y \lhd c \rhd z) \\
(1) &\approx (x \lhd c \rhd x) \cdot (y \lhd c \rhd z) \\
(4) &\approx x \cdot y \lhd c \rhd x \cdot z
\end{aligned}$$

6. $x \parallel y \approx y \parallel x$;

$$\begin{aligned}
& x \parallel y \\
(\text{CM1}) &\approx (x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x) + x \mid y \\
(\text{A1}), (\text{SC3}) &\approx (y \mathbin{⌊\!⌊} x + x \mathbin{⌊\!⌊} y) + y \mid x \\
(\text{CM1}) &\approx y \parallel x
\end{aligned}$$

7. $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$;

$$
\begin{aligned}
& (x \parallel y) \parallel z \\
\text{(CM1)} \approx\ & (x \parallel y) \mathbin{|\!\!\lfloor} z + z \mathbin{|\!\!\lfloor} (x \parallel y) + (x \parallel y) \mid z \\
\text{twice (CM1)} \approx\ & (x \mathbin{|\!\!\lfloor} y + y \mathbin{|\!\!\lfloor} x + x \mid y) \mathbin{|\!\!\lfloor} z + z \mathbin{|\!\!\lfloor} (x \parallel y) \\
& + (x \mathbin{|\!\!\lfloor} y + y \mathbin{|\!\!\lfloor} x + x \mid y) \mid z \\
\text{3 times (CM4) and (CM8)} \approx\ & (x \mathbin{|\!\!\lfloor} y) \mathbin{|\!\!\lfloor} z + (y \mathbin{|\!\!\lfloor} x) \mathbin{|\!\!\lfloor} z + (x \mid y) \mathbin{|\!\!\lfloor} z + z \mathbin{|\!\!\lfloor} (x \parallel y) \\
& + (x \mathbin{|\!\!\lfloor} y) \mid z + (y \mathbin{|\!\!\lfloor} x) \mid z + (x \mid y) \mid z \\
\text{twice (SC1) and (SC5)} \approx\ & x \mathbin{|\!\!\lfloor} (y \parallel z) + y \mathbin{|\!\!\lfloor} (x \parallel z) + (x \mid y) \mathbin{|\!\!\lfloor} z + z \mathbin{|\!\!\lfloor} (x \parallel y) \\
& + (x \mid z) \mathbin{|\!\!\lfloor} y + (y \mid z) \mathbin{|\!\!\lfloor} x + (x \mid y) \mid z \\
\text{(A1)} \approx\ & x \mathbin{|\!\!\lfloor} (y \parallel z) + y \mathbin{|\!\!\lfloor} (x \parallel z) + z \mathbin{|\!\!\lfloor} (x \parallel y) \\
& + (x \mid y) \mathbin{|\!\!\lfloor} z + (x \mid z) \mathbin{|\!\!\lfloor} y + (y \mid z) \mathbin{|\!\!\lfloor} x + (x \mid y) \mid z
\end{aligned}
$$

and

$$
\begin{aligned}
& x \parallel (y \parallel z) \\
\text{(6)} \approx\ & (y \parallel z) \parallel x \\
\text{above derivation} \approx\ & y \mathbin{|\!\!\lfloor} (z \parallel x) + z \mathbin{|\!\!\lfloor} (y \parallel x) + x \mathbin{|\!\!\lfloor} (y \parallel z) \\
& + (y \mid z) \mathbin{|\!\!\lfloor} x + (y \mid x) \mathbin{|\!\!\lfloor} z + (z \mid x) \mathbin{|\!\!\lfloor} y + (z \mid x) \mid y \\
\text{4 times (A1)} \approx\ & x \mathbin{|\!\!\lfloor} (y \parallel z) + y \mathbin{|\!\!\lfloor} (z \parallel x) + z \mathbin{|\!\!\lfloor} (y \parallel x) \\
& + (y \mid x) \mathbin{|\!\!\lfloor} z + (z \mid x) \mathbin{|\!\!\lfloor} y + (y \mid z) \mathbin{|\!\!\lfloor} x + (z \mid x) \mid y \\
\text{twice (6) and (SC3), (SC4)} \approx\ & x \mathbin{|\!\!\lfloor} (y \parallel z) + y \mathbin{|\!\!\lfloor} (x \parallel z) + z \mathbin{|\!\!\lfloor} (x \parallel y) \\
& + (x \mid y) \mathbin{|\!\!\lfloor} z + (x \mid z) \mathbin{|\!\!\lfloor} y + (y \mid z) \mathbin{|\!\!\lfloor} x + (x \mid y) \mid z
\end{aligned}
$$

8. $x \parallel \delta \approx x \cdot \delta$;

$$
\begin{aligned}
& x \parallel \delta \\
\text{(CM1)} \approx\ & x \mathbin{|\!\!\lfloor} \delta + \delta \mathbin{|\!\!\lfloor} x + \delta \mid x \\
\text{(SCD1), (CM2), (SCD2)} \approx\ & x \cdot \delta + \delta \cdot x + \delta \\
\text{(A7), (A6)} \approx\ & x \cdot \delta
\end{aligned}
$$

9. $x \parallel (y \cdot \delta) \approx (x \parallel y) \cdot \delta$;

$$
\begin{aligned}
& x \parallel (y \cdot \delta) \\
\text{(8)} \approx\ & x \parallel (y \parallel \delta) \\
\text{(7)} \approx\ & (x \parallel y) \parallel \delta \\
\text{(8)} \approx\ & (x \parallel y) \cdot \delta
\end{aligned}
$$

10. $\tau_I(\partial_H(x)) \approx \tau_{I \cup H}(\partial_H(x))$;

$$
\begin{aligned}
& \tau_I(\partial_H(x)) \\
\text{(T0)} \approx\ & \tau_I(\tau_\emptyset(\partial_H(x))) \\
\approx\ & \tau_I(\tau_{H \setminus H}(\partial_H(x))) \\
\text{(TDO)} \approx\ & \tau_I(\tau_H(\partial_H(x))) \\
\text{(TT)} \approx\ & \tau_{I \cup H}(\partial_H(x)))
\end{aligned}
$$

11. $\rho_R(\tau_I(\partial_H(x))) \approx \rho_{R_{I \cup H}}(\tau_I(\partial_H(x)))$;

$$
\begin{aligned}
& \rho_R(\tau_I(\partial_H(x))) \\
\text{(10)} \approx\ & \rho_R(\tau_{I \cup H}(\partial_H(x))) \\
\text{(RTO)} \approx\ & \rho_{R_{I \cup H}}(\tau_{I \cup H}(\partial_H(x))) \\
\text{(10)} \approx\ & \rho_{R_{I \cup H}}(\tau_I(\partial_H(x)))
\end{aligned}
$$

12. $\rho_R(\partial_H(x)) \approx \rho_{R_H}(\partial_H(x))$.

$$\begin{aligned} &\rho_R(\partial_H(x)) \\ &\quad (\mathrm{T0}) \approx \rho_R(\tau_\emptyset(\partial_H(x))) \\ &\quad (11) \approx \rho_{R_H}(\tau_\emptyset(\partial_H(x))) \\ &\quad (\mathrm{T0}) \approx \rho_{R_H}(\partial_H(x)) \end{aligned}$$

□

2.3 Timed μCRL

Several timed extensions have been proposed for different kinds of process algebras. For an overview of ACP extensions with time we refer to [9]. In [47] the language μCRL is extended with time, and in [55] a sound and complete axiomatization of timed μCRL is presented. In [58] some examples of specification and reasoning in timed μCRL are given.

In timed μCRL an extra sort *Time* is compulsory to denote time values. In [47] *Time* is defined as an abstract sort containing the total order relation $\leq$ and the least element $\mathbf{0}$, which were specified using conditional equational logic. In Chapter 6 we present a purely equational specification of *Time* that is sufficient to prove the correctness of the linearization procedure for timed μCRL.

A key feature of timed μCRL is that it can be expressed at which time certain action must take place. This is done using the *"at"-operator.* The process p ‹ t behaves like the process p, with the restriction that the first action of p must start at time t. Another key feature of timed μCRL is that it can be expressed that a process can delay till a certain time. The process $p + \delta$ ‹ t can certainly delay till time t, but can possibly delay longer, depending on p. Consequently, the process δ‹$\mathbf{0}$ can neither delay nor perform actions, and the process δ can delay for arbitrary long time, but cannot perform any action. It is assumed that a process that can delay till time t can also delay till any earlier moment, and a process that can perform a *first* action at time t can also delay till time t.

A number of other time-related operators were added to timed μCRL in order to enable a finite axiomatization of parallel composition. We refer to Chapter 6 for full description of the signature and the axioms of timed μCRL. We also note that all the identities derived in this chapter for μCRL can also be derived for timed μCRL in a similar way. The same holds for the theory of recursive specifications presented in Chapter 3.

Chapter 3

Recursive Specifications in μCRL

In this chapter we investigate the question how to relate different systems of equations in μCRL. First we consider simple non-parametric equations (the BPA case), which are interpreted in algebras, and then scale up to parametric equations in μCRL, which are interpreted in many sorted binding algebras.

3.1 Systems of Process Equations

We assume a fixed and infinite set $\mathsf{Procnames} = \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \dots\}$ of *process names* with type information associated to them. Formally speaking, process names in μCRL are second order variables. We extend the sort *Proc* of processes by allowing the process names in $P \subseteq \mathsf{Procnames}$ as variables of type $\overrightarrow{D} \to \mathit{Proc}$. The terms in the signature of μCRL extended with P are further called *(μCRL) process terms* and the set of all of them is denoted by $\mathit{Terms}(P)$. The *free data variables* in a process term are those not bound by $\sum_{d:D}$ occurrences. We write DVar for the set of all free and bound data variables that can occur in a term. We write $p(P, \overrightarrow{d})$ for a term p from $\mathit{Terms}(P)$, if all its free data variables are in $\overrightarrow{d}$.

Definition 3.1.1 (Process Equation). A *process equation* is an equation of the form $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = p_{\mathsf{X}}$, where X is a process name with a list of data parameters $\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}$, and p_{X} is a process term, in which only the data variables from $\overrightarrow{d_{\mathsf{X}}}$ may occur freely. We write $\mathit{rhs}(\mathsf{X})$ for p_{X}, $\mathit{pars}(\mathsf{X})$ for $\overrightarrow{d_{\mathsf{X}}}$, and $\mathit{type}(\mathsf{X})$ for $\overrightarrow{D_{\mathsf{X}}}$.

Definition 3.1.2 (System of Process Equations). Let $P \subseteq \mathsf{Procnames}$ be a finite set of process names such that each process name is uniquely typed. A (finite) non-empty set G of process equations with the right-hand sides from $\mathit{Terms}(P)$ is called a *(finite) system of process equations* if each process name in P occurs exactly once at the left. The set of process names (with types) that appear within G is denoted as $|G|$ (so, $|G| = P$). We use $rhs(\mathsf{X}, G)$, $pars(\mathsf{X}, G)$ and $type(\mathsf{X}, G)$ to refer to the corresponding parts of the equation for X in G.

Although the original definition of a μCRL specification allows to have the same process names with different types, we do not treat this possibility here as it would make the explanation only more long-winded.

Definition 3.1.3 (Process Definition). Let G be a finite system of process equations, X be a process name in it, and $\overrightarrow{t}$ be a data term vector of type $type(\mathsf{X}, G)$. Then the pair $(\mathsf{X}(\overrightarrow{t}), G)$ is called a *process definition* [1]. We use the abbreviation (X, G) for $(\mathsf{X}(pars(\mathsf{X}, G)), G)$.

Process definitions in μCRL comprise a restricted form of recursive applicative program schemes as defined in [31, 32]. The restrictions are that all unknowns (process names) have the same range *Proc* and that there are no functions from *Proc* to other sorts. On the other hand, process definitions extend recursive applicative program schemes with binders (because the sum operators of μCRL are binders), and therefore require a more refined approach for a formal treatment, such as binding algebras [94].

Example 3.1.4. All of $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{Y}, \mathsf{Y} = \mathsf{b} \cdot \mathsf{X}, \mathsf{Z} = \mathsf{X} \parallel \mathsf{Y}\}$, $G_2 = \{\mathsf{T}(n{:}Nat) = \mathsf{a}(even(n)) \cdot \mathsf{T}(S(n))\}$ and $G_3 = \{\mathsf{X}(b{:}Bool) = \mathsf{a}(b) \cdot \mathsf{X}(\neg b)\}$ with $even : Nat \to Bool$ as expected and $S : Nat \to Nat$ the successor function, are examples of systems of process equations. All of (X, G_1), (T, G_2), $(\mathsf{T}(m), G_2)$, $(\mathsf{X}(\mathbf{t}), G_3)$ and $(\mathsf{X}(b), G_3)$ are process definitions.

Definition 3.1.5 (Process Name Dependency). Process term q *directly depends* on process name X if this name occurs in q. Process name X *directly depends* on process name Y in a system of process equations G if $rhs(\mathsf{X}, G)$ directly depends on Y. Process term q *depends* on X in G if either it directly depends on it, or there is a sequence of process names $\mathsf{Y}_1, \ldots, \mathsf{Y}_n = \mathsf{X}$ such that q directly depends on Y_1 and for each $i < n$, Y_i directly depends on Y_{i+1}. Process name X *depends* on Y in G if $rhs(\mathsf{X}, G)$ depends on it.

3.2 Equivalence of BPA Systems

Recall that the signature of BPA (Basic Process Algebra, see, e.g., [10, 39]) contains two operations $+$ and $\cdot$, where $+$ models alternative composition and $\cdot$ sequential composition. We further omit brackets in repeated applications of $+$ and $\cdot$. The axioms of BPA are the axioms (A1)–(A5) from Table 2.2.

For terms t, u over the signature of BPA we write $\text{BPA} \vdash t \approx u$ if $t \approx u$ can be derived from BPA in equational calculus. Furthermore, let $\vec{x} = x_1, \ldots, x_n$ be a sequence of variables. Then we write $t(\vec{x})$ if all free variables of t are in $\vec{x}$. In this section we consider systems of (recursive) equations over the signature of BPA. As a convention for this section we shall use capital letters for the variables in such systems, in order to distinguish these from the variables in the BPA axioms.

[1] This terminology is syntax-oriented; the question whether $(\mathsf{X}(\overrightarrow{t}), G)$ really 'defines' a process is a model dependent one.

Let n fresh variables $\mathsf{X}_1, \ldots, \mathsf{X}_n = \vec{\mathsf{X}}$ and terms $p_1(\vec{\mathsf{X}}), \ldots, p_n(\vec{\mathsf{X}})$ over BPA be fixed. We consider the system of equations G, where

$$G = \{\mathsf{X}_i = p_i(\vec{\mathsf{X}}) \mid i = 1, \ldots, n\}$$

Let $\mathcal{M}$ be a model of BPA with domain M. Then $\vec{m} = (m_1, \ldots, m_n) \in M^n$ is a *solution of* G *in* $\mathcal{M}$ if for all $i = 1, \ldots, n$ and interpretation functions $\mathcal{I}$ satisfying $\mathcal{I}(\mathsf{X}_i) = m_i$,

$$\mathcal{M}, \mathcal{I} \models \mathsf{X}_i \approx p_i(\vec{\mathsf{X}}). \tag{3.1}$$

We further abbreviate (statements like) (3.1) to $\mathcal{M} \models m_i \approx p_i(\vec{m})$. In this case we say that m_i is a solution of (X_i, G) in $\mathcal{M}$. Finally, given G as above, i.e., $G = \{\mathsf{X}_i = p_i(\vec{\mathsf{X}}) \mid i = 1, \ldots, n\}$, we define for a term sequence $\vec{v} = v_1, \ldots, v_n$,

$$G(\vec{v}) \stackrel{\text{df}}{=} \bigwedge_{i=1}^{n} v_i \approx p_i(\vec{v}).$$

Thus the fact that $\vec{m}$ is a solution of G is denoted as $\mathcal{M} \models G(\vec{m})$.

We now turn to the preservation of solutions. Let $H = \{\mathsf{Y}_j = q_j(\vec{\mathsf{Y}}) \mid j = 1, \ldots, k\}$ be a system of k process equations over BPA such that $\vec{\mathsf{X}}$ and $\vec{\mathsf{Y}} = \mathsf{Y}_1, \ldots, \mathsf{Y}_k$ do not share any variables. The preservation of solutions refers to designated process definitions of G and H, usually (X_1, G) and (Y_1, H), respectively.

Definition 3.2.1 (Preservation of Solutions). We say that (X_1, G) *is preserved by* (Y_1, H), notation $(\mathsf{X}_1, G) \preceq (\mathsf{Y}_1, H)$, if in each model of BPA, each solution of (X_1, G) is also a solution of (Y_1, H). Formally, $(\mathsf{X}_1, G) \preceq (\mathsf{Y}_1, H)$ if in each model $\mathcal{M}$ of BPA, say with domain M,

$$\forall \vec{m} \in M^n (\mathcal{M} \models G(\vec{m}) \Rightarrow \exists \vec{n} \in M^k (\mathcal{M} \models H(\vec{n}) \wedge m_1 = n_1)).$$

Our aim is to syntactically characterize the notion of preservation of solutions. To this end we turn the equations in G to axioms. Let $\overline{G}$ refer to the setting where $\mathsf{X}_1, \ldots, \mathsf{X}_n = \vec{\mathsf{X}}$ are regarded as constants and the equations in G as additional axioms. (For example, the equation $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ becomes the identity $\mathsf{X} \approx \mathsf{a} \cdot \mathsf{X}$ in the signature of BPA extended with $|G|$.)

Definition 3.2.2 (Implication of Process Definitions). We say that (X_1, G) *implies* (Y_1, H), notation $(\mathsf{X}_1, G) \Rightarrow (\mathsf{Y}_1, H)$, if there exist closed terms $\vec{w} = w_1, \ldots, w_k$ from the set of closed terms in the signature of BPA extended with process names from $|G|$ (notation $CTerms(\Sigma_{\text{BPA}} \cup |G|)$) such that

$$\text{BPA} \cup \overline{G} \vdash \mathsf{X}_1 \approx w_1, \quad \text{and for all } j = 1, \ldots, k, \quad \text{BPA} \cup \overline{G} \vdash w_j \approx q_j[\vec{\mathsf{Y}} := \vec{w}].$$

We proceed to show that $\preceq$ is characterized by $\Rightarrow$, i.e., preservation of solutions can be derived in the *equational* calculus.

Theorem 3.2.3. *Let* (X_1, G) *and* (Y_1, H) *be process definitions over BPA. Then* $(\mathsf{X}_1, G) \preceq (\mathsf{Y}_1, H)$ *iff* $(\mathsf{X}_1, G) \Rightarrow (\mathsf{Y}_1, H)$.

Proof. **If:** We need to show that in every model of BPA solutions of (X_1, G) are also solutions of (Y_1, H). Let $\mathcal{M}$ be a model of BPA with the carrier set M. If (X_1, G) has no solutions in $\mathcal{M}$, we have nothing to prove. Otherwise, let $\vec{m} = (m_1, \ldots, m_n) \in M^n$ be a solution of G in $\mathcal{M}$. Let $\overline{\mathcal{M}}$ be $\mathcal{M}$ extended with constants $\overline{m}_1, \ldots, \overline{m}_n$. It is clear that $\overline{\mathcal{M}}$ is a model of BPA $\cup\, \overline{G}$ where $\mathsf{X}_1, \ldots, \mathsf{X}_n$ are interpreted as $\overline{m}_1, \ldots, \overline{m}_n$. From the assumption of the theorem, by Definition 3.2.2 and soundness of equational calculus, we get that there exist closed terms $\vec{w} = w_1, \ldots, w_k$ from $CTerms(\Sigma_{\mathrm{BPA}} \cup |G|)$, such that $[\![w_1]\!]_{\overline{\mathcal{M}}}, \ldots, [\![w_k]\!]_{\overline{\mathcal{M}}}$ is a solution of H in $\mathcal{M}$, and $m_1 = [\![w_1]\!]_{\overline{\mathcal{M}}}$.

Only if: We need to show that there exist closed terms $\vec{w} = w_1, \ldots, w_k$ from $CTerms(\Sigma_{\mathrm{BPA}} \cup |G|)$ such that certain identities are derivable from BPA $\cup\, \overline{G}$. If BPA $\cup\, \overline{G}$ has no non-empty models, then, by completeness of the equational calculus, we can derive any identity from BPA $\cup\, \overline{G}$, including the ones needed to show the statement of our theorem.

Otherwise, consider a non-empty model $\mathcal{M}$ of BPA $\cup\, \overline{G}$ with the carrier M and the vector of constants $\vec{\overline{m}} = (\overline{m}_1, \ldots, \overline{m}_n)$, with values $\vec{m} = (m_1, \ldots, m_n) \in M^n$, that are interpretations of $\mathsf{X}_1, \ldots, \mathsf{X}_n$. It is clear that $\vec{m}$ is a solution of G in $\mathcal{M}$. From the assumption of the theorem we know that for some $\vec{n} = (n_1, \ldots, n_k) \in M^k$,

$$\mathcal{M} \models H(\vec{n}) \wedge m_1 = n_1$$

If all elements of $\vec{n}$ can be represented as interpretations of closed terms from $CTerms(\Sigma_{\mathrm{BPA}} \cup |G|)$, then, by completeness of equational calculus, we obtain the statement of the theorem.

To show that such vector $\vec{n}$ exists, we consider the subalgebra $\widehat{\mathcal{M}}$ of $\mathcal{M}$ obtained by restricting M to $\widehat{M}$, which contains only the interpretations of closed terms in $CTerms(\Sigma_{\mathrm{BPA}} \cup |G|)$. This subalgebra is also a model of BPA $\cup\, \overline{G}$, because the set of models of any equational theory is closed under formation of subalgebra. Again from the assumption of the theorem we know that for some $\vec{n} = (n_1, \ldots, n_k) \in \widehat{M}^k$,

$$\widehat{\mathcal{M}} \models H(\vec{n}) \wedge m_1 = n_1$$

Due to the fact that all of the identities in the above formula are between closed terms, and the sets of operations of $\mathcal{M}$ and $\widehat{\mathcal{M}}$ coincide, the identities are also valid in $\mathcal{M}$. $\square$

Implication between process definitions induces the following equivalence between process definitions: $(\mathsf{X}_1, G) = (\mathsf{Y}_1, H)$ if $(\mathsf{X}_1, G) \Rightarrow (\mathsf{Y}_1, H)$ and $(\mathsf{Y}_1, H) \Rightarrow (\mathsf{X}_1, G)$. Evidently, this is an equivalence relation.

Some examples. If $G = \{\mathsf{X} = \mathsf{X}+\mathsf{a}+\mathsf{b}\}$ and $H = \{\mathsf{Y} = \mathsf{Y}+\mathsf{a}\}$, then $(\mathsf{X}, G) \Rightarrow (\mathsf{Y}, H)$ but not vice versa.

If $G = \{\mathsf{X}_1 = \mathsf{a} \cdot \mathsf{X}_2,\ \mathsf{X}_2 = \mathsf{b} \cdot \mathsf{X}_1\}$ and $H = \{\mathsf{Y} = \mathsf{a} \cdot \mathsf{b} \cdot \mathsf{Y}\}$, then $(\mathsf{X}_1, G) = (\mathsf{Y}, H)$. The implication from left to the right can be shown by choosing $w_{\mathsf{Y}} = \mathsf{X}_1$. The reverse direction can be shown by choosing $w_{\mathsf{X}_1} = \mathsf{Y}$ and $w_{\mathsf{X}_2} = \mathsf{b} \cdot \mathsf{Y}$.

The systems $G = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}\}$ and $H = \{\mathsf{Y} = \mathsf{a} \cdot \mathsf{Y} \cdot \mathsf{b}\}$ are incomparable: in the model with domain $\mathbb{Z}$, and with $+$ interpreted as maximum, $\cdot$ as addition, a as the value -1 and b as the value 1, there are no solutions for X and many for Y. The converse holds in case a is interpreted as 0 and b as 1.

If $G = \{\mathsf{X}_1 = \mathsf{a}+\mathsf{X}_1\cdot\mathsf{a},\ \mathsf{X}_2 = \mathsf{a}\cdot\mathsf{X}_2\}$ and $H = \{\mathsf{Y} = \mathsf{a}+\mathsf{Y}\cdot\mathsf{a}\}$, then $(\mathsf{X}_1, G) \Rightarrow (\mathsf{Y}, H)$, but the reverse implication does not hold. Consider the model where processes are trees with finite paths, but possibly infinitely branching, taken modulo bisimulation equivalence [45]. In this model (Y, H) has a solution, which is the class of trees representing the process $\sum_{i\in Nat} a^{i+1}$. But G has no solutions in this model, because of its second equation, which requires an infinite path. See [10, p. 33] and [7, p. 153] for more information about this counterexample.

3.3 Equivalence of Systems of Parameterized Equations

Consider a system G containing equations for $\mathsf{X}_1, \dots, \mathsf{X}_n$:

$$G = \{\mathsf{X}_i(\overrightarrow{d_{\mathsf{X}_i}{:}D_{\mathsf{X}_i}}) = p_i(\overrightarrow{\mathsf{X}}, \overrightarrow{d_{\mathsf{X}_i}}) \mid i = 1, \dots, n\}$$

Let the many sorted binding algebra $\mathcal{M}$ be a model of μCRL and the data types used in G. with domains P for processes, B for booleans and a family of other data domains D_i for each data domain in $\mathcal{M}$, and function domain $\mathcal{F}$. By D_{X_i} we denote the domain vector that corresponds to the type of parameters $\mathsf{D}_{\mathsf{X}_i}$ for process name X_i. A *solution* of G in $\mathcal{M}$ is a vector $(f_1, \dots, f_n)$ of functions $f_i : \overrightarrow{\mathsf{D}_{\mathsf{X}_i}} \to P$ from $\mathcal{F}$ such that for all $i = 1, \dots, n$ and interpretation functions $\mathcal{I}$ satisfying $\mathcal{I}(\mathsf{X}_i) = f_i$,

$$\mathcal{M}, \mathcal{I} \models \mathsf{X}_i(\overrightarrow{d_{\mathsf{X}_i}}) \approx p_i(\overrightarrow{\mathsf{X}}, \overrightarrow{d_{\mathsf{X}_i}}).$$

In this case for all $\overrightarrow{t}$ of type D_{X_i} $f_i(\overrightarrow{t_{\mathcal{M}}})$ is a solution of process definition $(\mathsf{X}_i(\overrightarrow{t}), G)$ in $\mathcal{M}$.

Given G as above, let $H = \{\mathsf{Y}_j(\overrightarrow{d_{\mathsf{Y}_j}{:}D_{\mathsf{Y}_j}}) = q_j(\overrightarrow{\mathsf{Y}}, \overrightarrow{d_{\mathsf{Y}_j}}) \mid j = 1, \dots, k\}$ be another system of process equations in the signature of μCRL and data domains of G.

Definition 3.3.1 (Preservation of Solutions). We say that $(\mathsf{X}_1(\overrightarrow{t}), G)$ *is preserved by* $(\mathsf{Y}_1(\overrightarrow{u}), H)$, notation $(\mathsf{X}_1(\overrightarrow{t}), G) \preceq (\mathsf{Y}_1(\overrightarrow{u}), H)$, if in each model $\mathcal{M}$ of μCRL and the data theories for both G and H,

$$\forall \overrightarrow{f} \in \mathcal{F}^n \Big((\forall i\ \mathcal{M} \models f_i(\overrightarrow{d_{\mathsf{X}_i}}) \approx t_i(\overrightarrow{f}, \overrightarrow{d_{\mathsf{X}_i}})) \implies$$
$$\exists \overrightarrow{g} \in \mathcal{F}^k (\forall j\ \mathcal{M} \models g_j(\overrightarrow{d_{\mathsf{Y}_j}}) \approx q_j(\overrightarrow{g}, \overrightarrow{d_{\mathsf{Y}_j}}) \wedge\ \mathcal{M} \models f_1(\overrightarrow{t}) \approx g_1(\overrightarrow{u})) \Big)$$

where the types of the functions f_i and g_j are $f_i : \overrightarrow{D_{\mathsf{X}_i}} \to P$ and $g_j : \overrightarrow{D_{\mathsf{Y}_j}} \to P$.

We now define the implication relation on process definitions in the setting of μCRL.

Definition 3.3.2 (Conditional Implication). We say that $(\mathsf{X}_1(\overrightarrow{t}), G)$ *conditionally implies* $(\mathsf{Y}_1(\overrightarrow{u}), H)$, notation $(\mathsf{X}_1(\overrightarrow{t}), G) \overset{cond}{\Rightarrow} (\mathsf{Y}_1(\overrightarrow{u}), H)$, if there exist terms

$w_j(\overrightarrow{\mathsf{X}}, \overrightarrow{d_{\mathsf{Y}_j}}) \in \mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{\mathit{DATA}} \cup |G|)$ with all free variables contained in $\overrightarrow{d_{\mathsf{Y}_j}}$ such that

$$\{\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}\} \vdash_{eBA} \mathsf{X}_1(\overrightarrow{t}) \approx w_1(\overrightarrow{u}),$$

and for all $j = 1, \ldots, k$,

$$\{\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}\} \vdash_{eBA} w_j(\overrightarrow{d_{\mathsf{Y}_j}}) \approx q_j[\overrightarrow{\mathsf{Y}} := \overrightarrow{w}](\overrightarrow{d_{\mathsf{Y}_j}}).$$

Here *DATA* represents the specification of the data types involved in both systems and in $\overrightarrow{t}$ and $\overrightarrow{u}$. Furthermore, $\overline{G}$ refers to the setting where the equations in G are considered to define additional axioms.

We continue with an example.

Example 3.3.3. Let $G = \{\mathsf{X}(b{:}\mathit{Bool}) = \mathsf{a}(b) \cdot \mathsf{X}(\neg b)\}$ and, with *Nat* a specification of the naturals, $H = \{\mathsf{Y}(n{:}\mathit{Nat}) = \mathsf{a}(\mathit{even}(n)) \cdot \mathsf{Y}(S(n))\}$. We show that

$$(\mathsf{X}(\mathbf{t}), G) \overset{cond}{\Rightarrow} (\mathsf{Y}(0), H)$$

by choosing $w(n) = \mathsf{X}(\mathit{even}(n))$. In this case we need to show that $\mathsf{X}(\mathbf{t}) \approx w(0)$ (this follows from $\mathit{even}(0) \approx \mathbf{t}$, which we assume to be derivable from *DATA*) and that $\mathsf{X}(\mathit{even}(n)) \approx \mathsf{a}(\mathit{even}(n)) \cdot \mathsf{X}(\mathit{even}(S(n)))$. This latter identity follows from $\mathsf{X}(b) \approx \mathsf{a}(b) \cdot \mathsf{X}(\neg b)$ and the necessarily derivable data identity $\mathit{even}(S(n)) \approx \neg \mathit{even}(n)$. If we assume the existence of a function $f : \mathit{Bool} \to \mathit{Nat}$, defined by $f(\mathbf{t}) \approx 0$ and $f(\mathbf{f}) \approx 1$ (where $\mathbf{f}$ stands for "false"), we can also prove that

$$(\mathsf{X}(b), G) \overset{cond}{\Rightarrow} (\mathsf{Y}(f(b)), H)$$

using the same term $w(n)$ and the data identities $\mathit{even}(f(b)) \approx b$ and $\mathit{even}(S(f(b))) \approx \neg b$, both of which seem reasonable. We do not have any of the reverse implications. Consider the model with carrier set *Nat*, in which $\mathsf{a}(b)$ is interpreted as 1, and sequential composition as $+$. Then $\mathsf{Y}(0)$ has many solutions, whereas $\mathsf{X}(\mathbf{t})$ has none.

Theorem 3.3.4. *Let* $(\mathsf{X}_1(\overrightarrow{t}), G)$ *and* $(\mathsf{Y}_1(\overrightarrow{u}), H)$ *be process definitions over μCRL. Then* $(\mathsf{X}_1(\overrightarrow{t}), G) \overset{cond}{\Rightarrow} (\mathsf{Y}_1(\overrightarrow{u}), H)$ *iff* $(\mathsf{X}_1(\overrightarrow{t}), G) \preceq (\mathsf{Y}_1(\overrightarrow{u}), H)$.

Proof. **If:** We need to show that in every model of μCRL and data theories for the data types used in G, H, $\overrightarrow{t}$, and $\overrightarrow{u}$, solutions of $(\mathsf{X}_1(\overrightarrow{t}), G)$ are also solutions of $(\mathsf{Y}_1(\overrightarrow{u}), H)$. Let $\mathcal{M}$ be such a model, with the carrier set M. If $(\mathsf{X}_1(\overrightarrow{t}), G)$ has no solutions in $\mathcal{M}$, we have nothing to prove. Otherwise, let $\overrightarrow{f} = (f_1, \ldots, f_n) \in \mathcal{F}^n$ be a solution of G in $\mathcal{M}$. Let $\overline{\mathcal{M}}$ be $\mathcal{M}$ where the set of operations is extended with functions $\overline{f}_1, \ldots, \overline{f}_n$. It is clear that $\overline{\mathcal{M}}$ is a model of $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$ where $\mathsf{X}_1, \ldots, \mathsf{X}_n$ are interpreted as $\overline{f}_1, \ldots, \overline{f}_n$. From the assumption of the theorem, by Definition 3.3.2 and soundness of equational calculus $\vdash_{eBA}$, we get that there exist terms $\overrightarrow{w} = w_1, \ldots, w_k$ from $\mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{\mathit{DATA}} \cup |G|)$ with all free variables of w_j contained in $\overrightarrow{d_{\mathsf{Y}_j}}$, such that $[\![w_1]\!]_{\overline{\mathcal{M}}}, \ldots, [\![w_k]\!]_{\overline{\mathcal{M}}}$ is a solution of H in $\mathcal{M}$, and $f_1(\overrightarrow{t}) \approx [\![w_1]\!]_{\overline{\mathcal{M}}}(\overrightarrow{u})$.

Only if: We need to show that there exist terms $\overrightarrow{w} = w_1, \ldots, w_k$ from $\mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{\mathit{DATA}} \cup |G|)$ with all free variables of w_j contained in $\overrightarrow{d_{\mathsf{Y}_j}}$. such that certain identities are derivable from $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$. If $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$ has no non-empty models, then, by completeness of calculus $\vdash_{eBA}$, we can derive any identity from $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$, including the ones needed to show the statement of our theorem.

Otherwise, consider a non-empty model $\mathcal{M}$ of $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$ with the carrier M and the vector of functions $\overrightarrow{\overline{f}} = (\overline{f}_1, \ldots, \overline{f}_n)$, with values $\overrightarrow{f} = (f_1, \ldots, f_n) \in \mathcal{F}^n$, that are interpretations of $\mathsf{X}_1, \ldots, \mathsf{X}_n$. It is clear that $\overrightarrow{f}$ is a solution of G in $\mathcal{M}$. From the assumption of the theorem we know that for some $\overrightarrow{g} = (g_1, \ldots, g_k) \in \mathcal{F}^k$,

$$\mathcal{M} \models g_j(\overrightarrow{d_{\mathsf{Y}_j}}) \approx q_j(\overrightarrow{g}, \overrightarrow{d_{\mathsf{Y}_j}}) \quad \text{and} \quad \mathcal{M} \models f_1(\overrightarrow{t}) \approx g_1(\overrightarrow{u}).$$

If all elements of $\overrightarrow{g}$ can be represented as interpretations of terms from $\mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{\mathit{DATA}} \cup |G|)$, then, by completeness of calculus $\vdash_{eBA}$, we obtain the statement of the theorem.

To show that such a vector $\overrightarrow{g}$ exists, we consider the subalgebra $\widehat{\mathcal{M}}$ of $\mathcal{M}$ obtained by restricting the domain of processes P to $\widehat{P}$, which contains only the interpretations of terms from $\mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{\mathit{DATA}} \cup |G|)$ of sort Proc. This subalgebra is also a model of $\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}$, because the set of models of any equational theory is closed under formation of subalgebra. Again from the assumption of the theorem we know that for some $\overrightarrow{g} = (g_1, \ldots, g_k) \in \widehat{M}^k$,

$$\mathcal{M} \models g_j(\overrightarrow{d_{\mathsf{Y}_j}}) \approx q_j(\overrightarrow{g}, \overrightarrow{d_{\mathsf{Y}_j}}) \quad \text{and} \quad \mathcal{M} \models f_1(\overrightarrow{t}) \approx g_1(\overrightarrow{u}).$$

Due to the fact that all of the identities in the above formula are between terms with no free variables of sort P, and the sets of operations of $\mathcal{M}$ and $\widehat{\mathcal{M}}$ coincide, the identities are also valid in $\mathcal{M}$. □

We state without proof:

Lemma 3.3.5 (Compositionality of Implications). *Let G_1 and G_2 be systems of process equations, and let the set H of process equations be such that $G_i \cup H$ is a system of process equations ($i = 1, 2$). If $G_1 \Rightarrow G_2$, then $G_1 \cup H \Rightarrow G_2 \cup H$, and if $G_1 \overset{cond}{\Rightarrow} G_2$, then $G_1 \cup H \overset{cond}{\Rightarrow} G_2 \cup H$.*

Definition 3.3.6 (Equivalence of Process Definitions). Process definition $(\mathsf{X}(\overrightarrow{t_1}), G_1)$ is *equivalent* to process definition $(\mathsf{Y}(\overrightarrow{t_2}), G_2)$ (notation $(\mathsf{X}(\overrightarrow{t_1}), G_1) = (\mathsf{Y}(\overrightarrow{t_2}), G_2)$) if both $(\mathsf{X}(\overrightarrow{t_1}), G_1) \Rightarrow (\mathsf{Y}(\overrightarrow{t_2}), G_2)$ and $(\mathsf{Y}(\overrightarrow{t_2}), G_2) \Rightarrow (\mathsf{X}(\overrightarrow{t_1}), G_1)$. Similarly, if $(\mathsf{X}(\mathit{pars}(\mathsf{X}, G_1)), G_1) = (\mathsf{Y}(\mathit{pars}(\mathsf{Y}, G_2)), G_2)$, we say that (X, G_1) is *equivalent* to (Y, G_2). *Conditional* equivalence (notation $\overset{cond}{=}$) is defined in the same way.

Finally, $G_1 = G_2$ if $|G_1| = |G_2|$ and $(\mathsf{X}, G_1) = (\mathsf{X}, G_2)$ for all $\mathsf{X} \in |G_1|$.

Note that on systems of process equations, the relations $=$ and $\overset{cond}{=}$ are equivalences, and the relations $\Rightarrow$ and $\overset{cond}{\Rightarrow}$ are preorders.

3.4 Equivalence in Inductive Domains

As mentioned in Chapter 2, the data types can be specified with the help of induction principles. In this case an (inductively defined) set of closed terms (normal forms) $T_n(D)$ represents all values of the data type D. In other words, we consider only such models $\mathcal{M}$ of μCRL, where the interpretations of $T_n(D)$ elements cover the entire carrier set for D.

For simplicity's sake we consider the case with only one inductively defined data type (the other case is an easy extension). We now define the *inductive implication* relation on process definitions in the setting of μCRL using the systems of equations G and H from the previous section.

Definition 3.4.1 (Inductive Implication). Let D be an inductively defined data type that is a component of $\mathit{pars}(\mathsf{X})$ vector. Let $T_n(D)$ be the set of normal forms of D. We say that $(\mathsf{X}_1(\overrightarrow{t}), G)$ *inductively implies* $(\mathsf{Y}_1(\overrightarrow{u}), H)$, notation

$$(\mathsf{X}_1(\overrightarrow{t}), G) \overset{ind}{\Rightarrow} (\mathsf{Y}_1(\overrightarrow{u}), H),$$

if there exist terms $w_j(\overrightarrow{\mathsf{X}}, \overrightarrow{d_{\mathsf{Y}_j}}) \in \mathit{Terms}(\Sigma_{\mu\mathrm{CRL}} \cup \Sigma_{DATA} \cup |G|)$ with all free variables contained in $\overrightarrow{d_{\mathsf{Y}_j}}$ such that

$$\{\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}\} \vdash_{eBA} \mathsf{X}_1(\overrightarrow{t}) \approx w_1(\overrightarrow{u}),$$

and for all $j = 1, \ldots, k$; and for all replacement functions σ_n that assign terms from $T_n(D)$ to the parameters of sort D, and do not change other variables,

$$\{\mu\mathrm{CRL} \cup \mathit{DATA} \cup \overline{G}\} \vdash_{eBA} w_j(\overrightarrow{\sigma_n(d_{\mathsf{Y}_j})}) \approx q_j[\overrightarrow{\mathsf{Y}} := \overrightarrow{w}](\overrightarrow{\sigma_n(d_{\mathsf{Y}_j})}).$$

Here DATA represents the specification of the data types involved in both systems and in $\overrightarrow{t}$ and $\overrightarrow{u}$. Furthermore, $\overline{G}$ refers to the setting where the equations in G are considered to define additional axioms.

Intuitively, this definition differs from Definition 3.3.2 in the fact that we derive the equations of H for normal forms of D only. The following theorem says that this is enough for the models with inductive interpretation of D.

Theorem 3.4.2. *Let $(\mathsf{X}_1(\overrightarrow{t}), G)$ and $(\mathsf{Y}_1(\overrightarrow{u}), H)$ be process definitions over μCRL. Then $(\mathsf{X}_1(\overrightarrow{t}), G) \overset{ind}{\Rightarrow} (\mathsf{Y}_1(\overrightarrow{u}), H)$ implies that in every model of μCRL with inductive carrier D, every solution of $(\mathsf{X}_1(\overrightarrow{t}), G)$ is a solution of $(\mathsf{Y}_1(\overrightarrow{u}), H)$.*

Proof. The proof goes along the same lines as the proof of the "if" part of Theorem 3.3.4. In the end we use the fact that the carrier set of D contains only the interpretations of elements in $T_n(D)$. Therefore the vector of functions for which the equations of H are valid for the representations of elements in $T_n(D)$ is actually a solution of H. $\square$

Similar to conditional and unconditional equality, *inductive equality* ($\overset{ind}{=}$) is defined as symmetric closure of inductive implication. It is clear that conditional implication implies inductive implication, so the transitive closure of both of them is inductive implication again. We illustrate the use of inductive equality with an example.

An example. Let *Nat* be a specification of the naturals comprising induction schemes (cf. e.g. [59]), and let G and H be the following systems of equations:

$$G = \left\{ \begin{array}{l} \mathsf{X}_1(n{:}Nat) = (\mathsf{a} \cdot \mathsf{X}_2(n-1) + \mathsf{X}_1(n-1)) \lhd n > 0 \rhd \mathsf{a}, \\ \mathsf{X}_2(n{:}Nat) = \mathsf{a} \cdot \mathsf{X}_2(n-1) \lhd n > 0 \rhd \mathsf{a} \end{array} \right\}$$
$$H = \left\{ \begin{array}{l} \mathsf{Y}_1(n{:}Nat) = (\mathsf{a} + \mathsf{Y}_1(n-1) \cdot \mathsf{a}) \lhd n > 0 \rhd \mathsf{a}, \\ \mathsf{Y}_2(n{:}Nat) = \mathsf{a} \cdot \mathsf{Y}_2(n-1) \lhd n > 0 \rhd \mathsf{a} \end{array} \right\}$$

We show that $(\mathsf{X}_k(n), G) \overset{ind}{=} (\mathsf{Y}_k(n), H)$ for $k = 1, 2$. For both implications $\overset{ind}{\Rightarrow}$ and $\overset{ind}{\Leftarrow}$ we choose the terms w_1 and w_2 to be trivial, namely, in the first case $w_1(n) = \mathsf{X}_1(n)$, $w_2(n) = \mathsf{X}_2(n)$, and in the second case $w_1(n) = \mathsf{Y}_1(n)$, $w_2(n) = \mathsf{Y}_2(n)$. The proofs then reduce to showing that $\{\mu\mathrm{CRL} \cup Nat \cup \overline{G}\} \vdash_{eBA} \mathsf{X}_1(n) \approx (\mathsf{a} + \mathsf{X}_1(n-1) \cdot \mathsf{a}) \lhd n > 0 \rhd \mathsf{a}$ and $\{\mu\mathrm{CRL} \cup Nat \cup \overline{H}\} \vdash_{eBA} \mathsf{Y}_1(n) \approx (\mathsf{a} \cdot \mathsf{Y}_2(n-1) + \mathsf{Y}_1(n-1)) \lhd n > 0 \rhd \mathsf{a}$.

First we show by induction on n that $\{\mu\mathrm{CRL} \cup NAT \cup \overline{G}\} \vdash_{eBA} \mathsf{a} \cdot \mathsf{X}_2(n) \approx \mathsf{X}_2(n) \cdot \mathsf{a}$. The case $n = 0$ is trivial. In the other case we get:

$$\mathsf{a} \cdot \mathsf{X}_2(n+1) \approx \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{X}_2(n) \overset{IH}{\approx} \mathsf{a} \cdot \mathsf{X}_2(n) \cdot \mathsf{a} \approx \mathsf{X}_2(n+1) \cdot \mathsf{a}$$

Similarly, $\{\mu\mathrm{CRL} \cup NAT \cup \overline{H}\} \vdash_{eBA} \mathsf{a} \cdot \mathsf{Y}_2(n) \approx \mathsf{Y}_2(n) \cdot \mathsf{a}$

Next, we show by induction on n that $\{\mu\mathrm{CRL} \cup NAT \cup \overline{G}\} \vdash_{eBA} \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{X}_1(n) \approx \mathsf{a} + \mathsf{X}_1(n) \cdot \mathsf{a}$. Again, for $n = 0$ we get $\mathsf{a} \cdot \mathsf{a} + \mathsf{a}$ in both sides. In the other case we get:

$$\mathsf{a} \cdot \mathsf{X}_2(n+1) + \mathsf{X}_1(n+1) \approx \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{X}_1(n)$$

and

$$\begin{aligned} \mathsf{a} + \mathsf{X}_1(n+1) \cdot \mathsf{a} &\approx \mathsf{a} + (\mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{X}_1(n)) \cdot \mathsf{a} \approx \mathsf{a} + \mathsf{a} \cdot \mathsf{X}_2(n) \cdot \mathsf{a} + \mathsf{X}_1(n) \cdot \mathsf{a} \\ &\approx (\mathsf{a} + \mathsf{X}_1(n) \cdot \mathsf{a}) + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{X}_2(n) \overset{IH}{\approx} \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{X}_1(n) + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{X}_2(n) \\ &\approx \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{a} \cdot \mathsf{X}_2(n) + \mathsf{X}_1(n) \end{aligned}$$

Next, we show that a similar identity is derivable from H, namely $\{\mu\mathrm{CRL} \cup NAT \cup \overline{H}\} \vdash_{eBA} \mathsf{a} \cdot \mathsf{Y}_2(n) + \mathsf{Y}_1(n) \approx \mathsf{a} + \mathsf{Y}_1(n) \cdot \mathsf{a}$. Again, the case $n = 0$ is trivial, and in the other case we have:

$$\mathsf{a} \cdot \mathsf{Y}_2(n+1) + \mathsf{Y}_1(n+1) \approx \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{Y}_2(n) + \mathsf{a} + \mathsf{Y}_1(n) \cdot \mathsf{a} \approx \mathsf{a} + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{Y}_2(n) + \mathsf{Y}_1(n) \cdot \mathsf{a}$$

and

$$\begin{aligned} \mathsf{a} + \mathsf{Y}_1(n+1) \cdot \mathsf{a} &\approx \mathsf{a} + (\mathsf{a} + \mathsf{Y}_1(n) \cdot \mathsf{a}) \cdot \mathsf{a} \overset{IH}{\approx} \mathsf{a} + (\mathsf{a} \cdot \mathsf{Y}_2(n) + \mathsf{Y}_1(n)) \cdot \mathsf{a} \\ &\approx \mathsf{a} + \mathsf{a} \cdot \mathsf{Y}_2(n) \cdot \mathsf{a} + \mathsf{Y}_1(n) \cdot \mathsf{a} \approx \mathsf{a} + \mathsf{a} \cdot \mathsf{a} \cdot \mathsf{Y}_2(n) + \mathsf{Y}_1(n) \cdot \mathsf{a} \end{aligned}$$

The last two identities imply the inductive equality we are proving.

It is important to note that the equation for Y_2 is not needed for the preservation of solutions. If H' is the system H with only the first equation, then $(\mathsf{Y}_1(n), H') \preceq (\mathsf{Y}_1(n), H)$. This is due to the fact that the equation for Y_2 has a solution in each model of μCRL and NAT, namely the function $f : Nat \to P$ such that $f(0) = \mathsf{a}$ and $f(n+1) = \mathsf{a} \cdot f(n)$. This differs with the necessity of the equation for Y_2 in the last example of Section 3.2.

3.5 Special Cases of System Equivalence

The following lemma shows that by applying a μCRL axiom to the right-hand side of an equation we get an equivalent system.

Lemma 3.5.1 (Axioms). *Let p_1, p_2 be process terms such that $p_1 \approx p_2$ is derivable. Let G be a system of process equations, and X be a process name in it such that p_1 is a subterm of rhs(X, G). Let G' consist of equations in G, except that in the equation defining X an occurrence of p_1 is replaced by p_2. Then $G = G'$.*

Proof. The statement of the theorem follows trivially from the fact that $p_1 \approx p_2$ is derivable. □

The following lemma shows that by replacing a subterm of the right-hand side of an equation by a fresh process name, and adding the equation for it, we get an equivalent process definition for each process name in the original system.

Lemma 3.5.2 (New Equation). *Let G be a system of process equations, and X be a process name in it. Let p be a subterm of rhs(X, G) with free data variables $d^1{:}D^1, \ldots, d^n{:}D^n = \overrightarrow{d{:}D}$ in it. Let Y be a process name, $\mathsf{Y} \notin |G|$. Let G' consist of equations in G, except that in the equation defining X an occurrence of p is replaced by $\mathsf{Y}(\overrightarrow{d})$, and the equation $\mathsf{Y}(\overrightarrow{d{:}D}) = p$ is added to G. Then for any $\mathsf{Z} \in |G|$ we have $(\mathsf{Z}, G) = (\mathsf{Z}, G')$.*

Proof. To prove that $(\mathsf{Z}, G) \Rightarrow (\mathsf{Z}, G')$ we take $w_{\mathsf{Z}}(pars(\mathsf{Z})) = \mathsf{Z}(pars(\mathsf{Z}))$ for all $\mathsf{Z} \in |G|$, and $w_{\mathsf{Y}} = p$. To prove the other direction we just take $w_{\mathsf{Z}}(pars(\mathsf{Z})) = \mathsf{Z}(pars(\mathsf{Z}))$ for all $\mathsf{Z} \in |G|$. □

The following lemma shows that under certain conditions we can substitute a process name by its right-hand side in the right-hand side of an equation. The condition says that we cannot use the same equation as both the target and the body of a replacement. For example, replacement of X in the right-hand side of $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ is not allowed.

Lemma 3.5.3 (Substitution). *Let G be a system of process equations, and X be a process name in it. Let $\mathsf{Y}(\overrightarrow{t})$ be a subterm of rhs(X, G) for some $\mathsf{Y} \neq \mathsf{X}$. Let G' consist of equations in G, except that in the equation defining X an occurrence of $\mathsf{Y}(\overrightarrow{t})$ is replaced by rhs$(\mathsf{Y}, G)[pars(\mathsf{Y}, G) := \overrightarrow{t}]$. Then we have that $G = G'$.*

Proof. In both directions we take the mappings w_{X} to be the identity mappings. □

The following lemma says that we can add dummy data parameters to a process equation, or remove such parameters.

Lemma 3.5.4 (Extra Parameters). *Let G be a system of process equations, and X be a process name in it with parameters $d^1, \ldots, d^n$. Suppose that d^i does not occur freely in rhs(X, G). Let G' be as G, but the process name X is replaced by X' and $pars(\mathsf{X}', G') = d^1, \ldots, d^{i-1}, d^{i+1}, \ldots, d^n$. Then for all $\mathsf{Y} \in |G| \wedge \mathsf{Y} \neq \mathsf{X}$ we have $(\mathsf{Y}, G) = (\mathsf{Y}, G')$, and $(\mathsf{X}(d^1, \ldots, d^n), G) = (\mathsf{X}'(d^1, \ldots, d^{i-1}, d^{i+1}, \ldots, d^n), G')$.*

Proof. In both directions we take the mappings w_{Y} (for $\mathsf{Y} \neq \mathsf{X}$) to be the identity mappings. In one direction $w_{\mathsf{X}'}(d^1, \ldots, d^{i-1}, d^{i+1}, \ldots, d^n) = \mathsf{X}(d^1, \ldots, d^n)$ and $w_{\mathsf{X}}(d^1, \ldots, d^n) = \mathsf{X}'(d^1, \ldots, d^{i-1}, d^{i+1}, \ldots, d^n)$. □

In many cases we are interested in a process definition (X, G) for a fixed process name X. The following lemma states that we can drop a defining equation for a process name $\mathsf{Y} \neq \mathsf{X}$, in cases when the X does not depend on Y, and Y does not depend on itself, and under the condition that the resulting set of equations will form a system of process equations (Definition 3.1.2).

Lemma 3.5.5 (Unreachable Equation). *Let G be a system of process equations, and X, Y be process names in it such that X does not depend on Y, and Y does not depend on itself. Let G' contain all equations in G except the defining equation for Y. If G' is a system of process equations, then we have $(\mathsf{X}, G) = (\mathsf{X}, G')$.*

Proof. In the direction from left to the right we use the identity mapping for w_{Z}. In the reverse direction we use the same mapping, but $w_{\mathsf{Y}} = \mathit{rhs}(\mathsf{Y}, G)$. □

3.6 Guardedness

In this thesis we use a slightly different notion of guardedness than the one in [52].

Definition 3.6.1. An occurrence of a process name X in a process term p is *completely guarded* if there is a subterm p' of p of the form $q \cdot p''$ containing this occurrence of X, where q is a process term containing no process names.

A process term is called *completely guarded* if every occurrence of a process name in it is completely guarded. Note that a term that contains no process names is completely guarded.

A system of process equations G is *completely guarded* if for any $\mathsf{X} \in |G|$, $\mathit{rhs}(\mathsf{X}, G)$ is a completely guarded term.

Definition 3.6.2. A process definition (X, G) is *(unconditionally) guarded* if there is a process definition (X', G') such that G' is a completely guarded system of process equations, and $(\mathsf{X}, G) = (\mathsf{X}', G')$.

Definition 3.6.3. Let G be a system of process equations. A *Process Name Unguarded-Dependency Graph (PNUDG)* is an oriented graph with the set of nodes $|G|$, and edges defined as follows: $\mathsf{X} \to \mathsf{Y}$ belongs to the graph if Y is not completely guarded in $\mathit{rhs}(\mathsf{X}, G)$.

Lemma 3.6.4. *If the PNUDG of a finite system of process equations G is acyclic, then G is guarded.*

Proof. Given a system G we replace each unguarded occurrence of a process name by its right-hand side. By Lemma 3.5.3 we get an equivalent system. Due to the fact that PNUDG is acyclic, we need to perform the replacement only finitely many times, and after that we get a completely guarded system. □

The following example shows that the converse of Lemma 3.6.4 does not hold.

Example 3.6.5. The system G consisting of one equation $\mathsf{X} = \mathsf{X} \lhd \mathbf{f} \rhd \delta$ is guarded, but its PNUDG contains the cycle $\mathsf{X} \to \mathsf{X}$.

3.7 Relation to RDP, RSP and CL-RSP

Solutions of process definitions are important within process algebra. The treatise as is given here, by comparing systems of equations by considering the preservation of solutions, is new. However, both approaches are strongly related and can be used fruitfully in combination. For instance, an important principle of classical process algebra is that every system of equations has a solution. This principle is called the Recursive Definition Principle (RDP). Another traditional principle is that every guarded system of equations has at most one solution. This principle is called Recursive Specification Principle (RSP). In the setting with data the principle CL-RSP (cf. [20]) is used in place of RSP. CL-RSP holds in a model of μCRL if every convergent LPE has at most one solution in this model. Convergence of an LPE means that in cannot perform infinite sequences of internal (τ) actions (cf. [20] for precise definition).

The combination of the principles RDP and (CL-)RSP can be used to prove that sets of solutions of process definitions are equal, although only implication is shown, in the following way.

Theorem 3.7.1. *If for some process definition $(\mathsf{X}(\overrightarrow{t_0}), G)$ there is a system L containing a single convergent LPE Z such that $(\mathsf{X}(\overrightarrow{t_0}), G) \overset{ind}{\Rightarrow} (\mathsf{Z}(\overrightarrow{u_0}), L)$ for some term vector $\overrightarrow{u_0}$, then in the models of μCRL, where both RDP and CL-RSP hold, both of the process definitions have the same unique solution.*

Proof. Consider a model $\mathcal{M}$ of μCRL where both RDP and CL-RSP hold. According to RDP, both process definitions have at least one solution in $\mathcal{M}$. In addition to that, according to CL-RSP, the process definition $(\mathsf{Z}(\overrightarrow{u_0}), L)$ has exactly one solution in $\mathcal{M}$. According to Section 3.4 every solution of $(\mathsf{X}(\overrightarrow{t_0}), G)$ is a solution of $(\mathsf{Z}(\overrightarrow{u_0}), L)$. This implies that $(\mathsf{X}(\overrightarrow{t_0}), G)$ cannot have more than one solution in $\mathcal{M}$, therefore the solution of $(\mathsf{Z}(\overrightarrow{u_0}), L)$ in $\mathcal{M}$ is the unique solution of $(\mathsf{X}(\overrightarrow{t_0}), G)$ in $\mathcal{M}$. $\square$

Chapter 4

Linearization in Parallel pCRL

4.1 μCRL and Parallel pCRL specifications

We restrict to the μCRL specifications that do not contain left merge ($\lfloor\!\lfloor$) and communication ($|$) explicitly. These operators were introduced to allow the finite axiomatization of parallel composition ($\|$) in the bisimulation setting, and they are hardly used explicitly in μCRL specifications. These operations can be easily eliminated from closed process terms, but their elimination from a μCRL specification requires several additional transformation steps.

We consider systems of process equations with the right-hand sides from the following subset of μCRL terms

$$p ::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p{+}p \mid p{\cdot}p \mid p\|p \mid \sum_{d:D} p \mid p{\lhd}c{\rhd}p \mid \partial_H(p) \mid \tau_I(p) \mid \rho_R(p) \quad (4.1)$$

The combination of the given data specification with a process definition $(\mathsf{X}(\overrightarrow{t}), G)$ of process equations determines a μCRL *specification* in the sense as defined in [53]. Such a specification depends on a finite subset **act** of *ActLab* and on **comm**, an enumeration of γ restricted to the labels in **act**. So a finite system G implicitly describes a finitary based language.

Furthermore, the *eq* functions for the data sorts we assume to have the following properties:

$$\{DATA, eq(d,e) \approx \mathbf{t}\} \vdash d \approx e \quad \text{and} \quad \{DATA, d \approx e\} \vdash eq(d,e) \approx \mathbf{t}$$

This allows us to denote terms that test for equality, which we use in the sequel, and these denotations will have the intended meaning. All data sorts that are introduced during the linearization have *eq* functions satisfying these properties.

The problem of linearization of a μCRL specification defined by $(\mathsf{X}(\overrightarrow{t}), G)$ consists of the generation of a new μCRL specification which

- depends on the same set of actions and communication function,
- contains all data definitions of the original one, and, possibly, definitions of auxiliary data types,
- is defined by $(\mathsf{Z}(\mathrm{m}_{\mathsf{X}}(\overrightarrow{t})), L)$, where L contains exactly one process equation for Z in linear form (defined later), and m_{X} is a mapping from $pars(\mathsf{X}, G)$ to $pars(\mathsf{Z}, L)$,

such that $(\mathsf{X}(\overrightarrow{t}), G) \overset{cond}{\Rightarrow} (\mathsf{Z}(\mathrm{m}_{\mathsf{X}}(\overrightarrow{t})), L)$.

It is not possible to linearize a μCRL specification which is unguarded. In this thesis we describe the linearization procedures for specifications, where the system of the equations has acyclic PNUDG. (Conditionally) guarded systems with cyclic PNUDG are not treated in this thesis. We note that in some cases cycles can be removed, for example because they are not reachable, or using properties of data types (cf. [60]). The elimination of cycles involves reachability analysis, which relies on theorem proving techniques for the data types used in a particular specification, and therefore is not treated here.

Parallel pCRL

We define (parallel) pCRL processes as a subset of μCRL processes. This subset is large enough to express many practical systems, and it requires a relatively simple linearization procedure.

Definition 4.1.1 (pCRL (Process) Equations). Let G be a system of process equations. A process term in $\mathit{Terms}(|G|)$ is called a *pCRL process term* in G if it has the syntax

$$p ::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p + p \mid p \cdot p \mid \sum_{d:D} p \mid p \lhd c \rhd p \tag{4.2}$$

and can directly depend only on process names whose right-hand sides are also pCRL process terms. A process name is called a *pCRL process name* if its right-hand side is a pCRL process term.

Definition 4.1.2 (Parallel pCRL (Process) Equation). Let G be a system of process equations. A process term in $\mathit{Terms}(|G|)$ is called a *parallel pCRL process term* in G if it has the syntax

$$q ::= \mathsf{Y}(\overrightarrow{t}) \mid q \parallel q \mid \partial_H(q) \mid \tau_I(q) \mid \rho_R(q) \tag{4.3}$$

and directly depends only on process names whose right-hand side are pCRL or parallel pCRL process terms. It is called a *parallel pCRL process name* if its right-hand side is a parallel pCRL process term.

Example 4.1.3. Referring to G_1 and G_2 as defined in Example 3.1.4, $\mathsf{X} + \mathsf{a}$ is a pCRL process term in G_1, and X, $\mathsf{X} \parallel \mathsf{X}$ and $\mathsf{X} \parallel \mathsf{Y}$ are parallel pCRL process terms in G_1. Furthermore, $\mathsf{T}(S(n))$ with n a variable of sort Nat and $\mathsf{a}(\mathit{even}(0)) \cdot \mathsf{T}(0)$ are pCRL process terms in G_2. Finally, $\mathsf{X} \parallel \mathsf{a}$ is *not* a (parallel) pCRL process term in G_1.

In the following definition we define what a parallel pCRL process definition is. For this definition we assume that we have a μCRL specification that is Statically Semantically Correct (cf. [53]), that is, in which the data types, actions, communication functions and processes are all well-defined. The first two restrictions posed in the definition below distinguish parallel pCRL as a subset of μCRL. The third one is present to disallow parallel process names on which the initial process name does not depend, and to exclude the presence of certain independent equations in a system. This is not a severe restriction and it simplifies the algorithm presented in Section 4.5.

Definition 4.1.4 (Parallel pCRL Process Definition). Let G be a finite system of process equations, and (X, G) be a process definition. (X, G) is called a *parallel pCRL process definition* if X is a (parallel) pCRL process name, and

- all of the process names in G are either pCRL or parallel pCRL process names;
- no parallel pCRL process name depends on itself;
- process name X depends on all parallel pCRL process names in G, but not on itself.

It is called a *pCRL system of process equations* if all process names in it are pCRL process names.

It follows from Definitions 4.1.4 and 4.1.2 that for every (parallel) pCRL process definition (X, G), either X is a pCRL process name, or it depends on a pCRL process name in G.

Example 4.1.5. Referring to G_1 as defined in Example 3.1.4, (Z, G_1) is a parallel pCRL process definition, but (X, G_1) is not.

4.1.1 Normal Forms

Below we define several normal forms for systems of process equations in parallel pCRL and μCRL, namely Extended Greibach Normal Form (EGNF), Parallel Extended Greibach Normal Form (PEGNF) and similar forms. A system is said to be in one of these forms if all of its equations are in the respective form.

From this point on we assume that $\mathsf{a}(\overrightarrow{t})$ with possible indices can also be an abbreviation for τ. This is done to make the normal form representations more concise.

Definition 4.1.6 (pre-GNF Forms). A μCRL process equation is in *pre-EGNF* if it is of the form:

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d, e_i}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta$$

where $p_i(\overrightarrow{d, e_i})$ are terms of the following syntax:

$$\begin{aligned} p &::= q \mid q \cdot \delta \\ q &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid \mathsf{a}(\overrightarrow{t}) \cdot q \mid \mathsf{Y}(\overrightarrow{t}) \cdot q \end{aligned} \tag{4.4}$$

A μCRL process equation is in *pre-PEGNF* if it is of the same form as above, but the terms $p_i(\overrightarrow{d,e_i})$ have the following syntax:

$$\begin{aligned} &p ::= q \mid q \cdot \delta \\ &q ::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid q \cdot q \mid q \parallel q \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \mid \rho_R(\tau_I(\partial_H(q \parallel q))) \end{aligned} \tag{4.5}$$

Definition 4.1.7 (GNF Forms). A μCRL process equation is in *EGNF* if it is of the form:

$$\begin{aligned} \mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta \\ &+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta \end{aligned} \tag{4.6}$$

where I and J are disjoint, and all $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$p ::= \mathsf{Y}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \cdot p \tag{4.7}$$

A μCRL process equation is in *PEGNF* if it is of the same form as above, but the terms $p_i(\overrightarrow{d,e_i})$ have the syntax (4.5).

A μCRL process equation is in *post-PEGNF* if it is of the same form as above, but the terms $p_i(\overrightarrow{d,e_i})$ have the following syntax:

$$p ::= \mathsf{Y}(\overrightarrow{t}) \mid p \cdot p \mid p \parallel p \mid \rho_R(\tau_I(\partial_H(p \parallel p))) \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \tag{4.8}$$

A μCRL process equation is called *Linear Process Equation (LPE)* if it is of the same form as above, but the terms $p_i(\overrightarrow{d,e_i})$ are recursive calls of the form $\mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d,e_i}))$ for some function vectors $\overrightarrow{g_i}$.

Note (Sum Notation). Apart from functions $\sum_{d:D} p$ that are included in the syntax of process terms, we use the following abbreviations. Expression $\sum_{\overrightarrow{d:D}}$ is an abbreviation for $\sum_{d^1:D^1} \cdots \sum_{d^n:D^n}$. In case $n = 0$, $\sum_{\overrightarrow{d:D}} p$ is an abbreviation for p. Expression $\sum_{i\in I} p_i$, where I is a finite set, is an abbreviation for $p_{i_1} + \cdots + p_{i_n}$ such that $\{i_1, \ldots, i_n\} = I$. In case $I = \emptyset$, $\sum_{i\in I} p_i$ is an abbreviation for δ.

Note (Conditions). As follows from the above definition, any process equation in (pre-)(post-)PEGNF must have a condition in each summand. However, this is not a necessary restriction. In case a summand q does not have a condition, it is an abbreviation for $q \lhd \mathbf{t} \rhd \delta$.

We also mention here that pre-(P)EGNF could be achieved by an algorithm similar to the one presented in Proposition 7.2 of [31]. There it is proved that every system of equations can be transformed to a *quasi-uniform* one by the introduction of new variables. In a quasi-uniform system each equation has at most one function symbol (in our case one function symbol of sort *Proc*) in the right hand side, which means that every such system is in pre-(P)EGNF. In our case such an algorithm would generate a lot more additional equations than necessary, many of which would become unreachable after performing the transformation in Subsection 4.2.4.

4.2 Transformation to Extended Greibach Normal Form

As the input for the linearization procedure, in this chapter, we take a (parallel) pCRL process definition (X, G) such that PNUDG of G is acyclic. The system of process equations G can be partitioned in two parts: G_1 and G_2, where G_1 has pCRL equations, and G_2 parallel pCRL equations. G_2 can be empty, in which case X is a pCRL process name. Otherwise X is a parallel pCRL process name.

In this section we transform G_1 into a system of process equations G_1' in EGNF. The resulting system will contain process equations for all process names in $|G_1|$ with the same names and types of data parameters involved, as well as, possibly, other process equations. After that we need to linearize the process definition (X, G'), where $G' = G_1' \cup G_2$.

4.2.1 Preprocessing

We first transform G_1 into G_1^1. This can be seen as a preprocessing step that possibly renames bound data variables. For instance $\sum_{d:D}((\sum_{d:E} \mathsf{a}(d)) \cdot \mathsf{b}(d))$ is replaced by $\sum_{d:D}((\sum_{e:E} \mathsf{a}(e)) \cdot \mathsf{b}(d))$, where e is a fresh variable. We replace each equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = p_\mathsf{X}$ in G_1 with the equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = S_0(\{\overrightarrow{d_\mathsf{X}}\}, p_\mathsf{X})$, where $S_0 : DVar \times Terms(|G_1|) \to Terms(|G_1|)$ is defined in the following way:

$$S_0(S, f(p^1, \ldots, p^n)) \to f(S_0(S, p^1), \ldots, S_0(S, p^n)) \text{ if } f \text{ is not } \sum_{d:D}$$

$$S_0\Big(S, \sum_{d:D} p\Big) \to \begin{cases} \sum_{d:D} S_0(S \cup \{d\}, p) & \text{if } d \notin S \\ \sum_{e:D} S_0(S \cup \{e\}, p[d := e]) & \text{if } d \in S \end{cases}$$

where e is a fresh variable.

Proposition 4.2.1. *Let G_1^1 be the result of applying the preprocessing to G_1. Then $G_1^1 = G_1$.*

Proof. The statement follows from Lemma 3.5.1 if we apply axiom (SUM2). □

As can easily be seen, the preprocessing step does not increase the size or the number of equations in the system.

4.2.2 Reduction by Simple Rewriting

By applying term rewriting we get an equivalent set of process equations to the given one, but with terms in right-hand sides in the more restricted form as presented in Table 4.1.

The rewrite rules that we apply to the right-hand sides of the equations are listed in Table 4.2. The symbols $\sum_{d:D}$ are treated in this rewrite system as function symbols, not as binders. This is justified by the fact that we have renamed all nested bound variables, which allows the use of first order term rewriting. We call the function

$$
\begin{aligned}
p &::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{X}(\overrightarrow{t}) \mid p_1 \cdot p \mid p_2 + p_2 \mid p_3 \lhd c \rhd \delta \mid \sum_{d:D} p_4 \\
p_1 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p_1 \cdot p \mid p_2 + p_2 \\
p_2 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p_1 \cdot p \mid p_2 + p_2 \mid p_3 \lhd c \rhd \delta \mid \sum_{d:D} p_4 \\
p_3 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p_1 \cdot p \\
p_4 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p_1 \cdot p \mid p_3 \lhd c \rhd \delta \mid \sum_{d:D} p_4
\end{aligned}
$$

Table 4.1: Syntax of terms after simple rewriting.

$$
\begin{aligned}
x + \delta &\to x && \text{(RA6)} \\
\delta \cdot x &\to \delta && \text{(RA7)} \\
\Big(\sum_{d:D} x\Big) \cdot y &\to \sum_{d:D} (x \cdot y) && \text{(RSUM5)} \\
(x \lhd c \rhd \delta) \cdot y &\to (x \cdot y) \lhd c \rhd \delta && \text{(RCOND6)} \\
\sum_{d:D} \delta &\to \delta && \text{(RSUM1}'\text{)} \\
\sum_{d:D} (x + y) &\to \sum_{d:D} x + \sum_{d:D} y && \text{(RSUM4)} \\
\delta \lhd c \rhd \delta &\to \delta && \text{(RCOND0}'\text{)} \\
(x + y) \lhd c \rhd \delta &\to x \lhd c \rhd \delta + y \lhd c \rhd \delta && \text{(RCOND7)} \\
\Big(\sum_{d:D} x\Big) \lhd c \rhd \delta &\to \sum_{d:D} x \lhd c \rhd \delta && \text{(RSUM12)} \\
(x \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta &\to x \lhd c_1 \wedge c_2 \rhd \delta && \text{(RCOND4)}
\end{aligned}
$$

Table 4.2: Rewrite rules defining *rewr*

induced by the rewrite rules *rewr* : $\mathit{Terms}(|G|) \to \mathit{Terms}(|G|)$ for a given system of process equations G.

Before applying the rewriting we eliminate all terms of the form $\_ \lhd \_ \rhd \_$ with the third argument being different from δ with the following rule:

$$y \not\equiv \delta \implies x \lhd c \rhd y \to x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta \qquad \text{(RCOND3)}$$

The rewriting is performed modulo the following rules:

$$
\begin{aligned}
x + y &\approx y + x \\
x + (y + z) &\approx (x + y) + z \\
(x \cdot y) \cdot z &\approx x \cdot (y \cdot z)
\end{aligned}
$$

The optimization rules presented in Table 4.3 are not needed to get the desired restricted syntactic form, but can be used to simplify the terms. They could be applied with higher priority than the rules in Table 4.2 to achieve possible reductions. Note that the rule (RSCA′) could lead to optimizations only in cases when x is completely guarded, and y or z are not.

$x + x \to x$	(RA3)
$x \lhd c \rhd x \to x$	(RCOND0)
$x \lhd \mathbf{t} \rhd y \to x$	(RCOND1)
$x \lhd \mathbf{f} \rhd y \to y$	(RCOND2)
$x \lhd c_1 \rhd \delta + x \lhd c_2 \rhd \delta \to x \lhd c_1 \vee c_2 \rhd \delta$	(RCOND5)
$(x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \to x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2$	(RSCA)
$x \cdot (y \lhd c \rhd z) \to x \cdot y \lhd c \rhd x \cdot z$	(RSCA′)

Table 4.3: Optimization rules.

Proposition 4.2.2. *The commutative/associative term rewriting system of Table 4.2 is terminating.*

Proof. We can transform this commutative/associative term rewriting system into a normal one by adding the rule $\delta + x \to \delta$ and directing the associativity axioms. Termination of the obtained system can be proved by constructing the recursive path ordering (RPO) for the following order on the operations: $\_ \lhd c \rhd \_ > \cdot > \_ \lhd c \rhd \delta > \sum > +$. □

Lemma 4.2.3. *For any process term p not containing $p_1 \lhd c \rhd p_2$, where $p_2 \not\equiv \delta$, we have that $rewr(p)$ has the syntax defined in Table 4.1.*

Proof. Let $q = rewr(p)$. It can be seen from the rewrite rules that they preserve the syntax in Definition 4.1.1. Suppose q does not satisfy the syntax defined in Table 4.1. The following possibilities exist, and all of them imply that q is reducible.

- $q = \delta + p_1$. Can be reduced by (RA6).
- $q = \delta \cdot p_1$. Can be reduced by (RA7).
- $q = (\sum_{d:D} p_1) \cdot p_2$. Can be reduced by (RSUM5).
- $q = (p_1 \lhd c \rhd \delta) \cdot p_2$. Can be reduced by (RCOND6).
- $q = \sum_{d:D} \delta$. Can be reduced by (RSUM1′).
- $q = \sum_{d:D}(p_1 + p_2)$. Can be reduced by (RSUM4).
- $q = \delta \lhd c \rhd \delta$. Can be reduced by (RCOND0′).
- $q = (p_1 + p_2) \lhd c \rhd \delta$. Can be reduced by (RCOND7).
- $q = (\sum_{d:D} p_1) \lhd c \rhd \delta$. Can be reduced by (RSUM12).
- $q = (p_1 \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta$. Can be reduced by (RCOND4).

□

Proposition 4.2.4. *Let G_1^2 be the result of applying the rewriting to G_1^1. Then $G_1^2 = G_1^1$.*

Proof. Taking into account that G_1^1 does not contain nested occurrences of bound variables, each rewrite rule is a consequence of the axioms of μCRL(cf. Lemma 2.2.6). By Lemma 3.5.1 we get $G_1^2 = G_1^1$. □

As the result of applying simple rewriting the number of equations obviously remains the same. The process terms may grow with a constant factor, but the number of occurrences of action labels and process names does not increase. The data terms and the number of their occurrences may grow with a constant factor, too.

4.2.3 Adding New Process Equations

In this step we reduce the complexity of terms in the right-hand sides of the G_1^2 equations even further by the introduction of new process equations. In some cases we take a subterm of a right-hand side and substitute it by a fresh process name parameterized by (at least) all free variables that appear in that subterm. As the result we get a system of process equations G_1^3 with equations in pre-EGNF. Such a transformation can be performed for all equations $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = p_{\mathsf{X}}$ by replacing them with $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = S_1(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}, p_{\mathsf{X}})$.

$$
\begin{aligned}
S_1(S, \mathsf{a}(\vec{t})) &\to \mathsf{a}(\vec{t})\\
S_1(S, \delta) &\to \delta\\
S_1(S, \mathsf{X}(\vec{t})) &\to \mathsf{X}(\vec{t})\\
S_1(S, p_1 \cdot p_2) &\to S_2(S, p_1 \cdot p_2)\\
S_1(S, p_1 + p_2) &\to S_1(S, p_1) + S_1(S, p_2)\\
S_1(S, p \lhd c \rhd \delta) &\to S_2(S, p) \lhd c \rhd \delta\\
S_1\Big(S, \sum_{d:D} p\Big) &\to \sum_{d:D} S_1(S \mathbin{\&} d{:}D, p)
\end{aligned}
$$

$$
\begin{aligned}
S_2(S, \mathsf{a}(\vec{t})) &\to \mathsf{a}(\vec{t})\\
S_2(S, \delta) &\to \delta\\
S_2(S, \mathsf{X}(\vec{t})) &\to \mathsf{X}(\vec{t})\\
S_2(S, p_1 \cdot p_2) &\to S_2(S, p_1) \cdot S_2(S, p_2)\\
S_2(S, p_1 + p_2) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\\
&\qquad \mathit{add}(\mathsf{Y}(S) = S_1(S, p_1 + p_2))\\
S_2(S, p \lhd c \rhd \delta) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\\
&\qquad \mathit{add}(\mathsf{Y}(S) = S_1(S, p \lhd c \rhd \delta))\\
S_2\Big(S, \sum_{d:D} p\Big) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\\
&\qquad \mathit{add}\Big(\mathsf{Y}(S) = S_1\big(S, \textstyle\sum_{d:D} p\big)\Big)
\end{aligned}
$$

Table 4.4: Transformations S_1 and S_2.

The transformations S_1 and S_2 are defined in the Table 4.4, where *fresh_var* represents a fresh process name, and *add* represents addition of the equation to the resulting system. Formally, S_1 and S_2 induce operations $\hat{S}_1$ and $\hat{S}_2$ that operate on sets of equations and are defined in the expected way (those operations actually transform the system of recursive equations).

The transformation S_1 distributes over all operations that preserve the form of right-hand side of equations in pre-EGNF. These are all operations except for sequential compositions, for which we apply the transformation S_2. The transformation S_2 distributes over all operations that preserve the syntax (4.4). These are all operations except for alternative composition, sums and conditions, for which we introduce new equations, as preserving them would break pre-EGNF. In the following we provide a simple example of the transformation.

Example 4.2.5. Let $G = \{\mathsf{X}(d{:}D) = \mathsf{a}(d) \cdot (\mathsf{b}(d) + \mathsf{X}(f(d)))\}$ be a given system of process equations. After applying the transformation S_1 we get the system $G' = \{\mathsf{X}(d{:}D) = \mathsf{a}(d) \cdot \mathsf{Y}(d), \mathsf{Y}(d{:}D) = \mathsf{b}(d) + \mathsf{X}(f(d))\}$ which is in pre-EGNF.

Proposition 4.2.6. *The functions S_1 and S_2 are well-defined.*

Proof. Using the order on the operations $S_1 > +, S_1 > \sum, S_2 > \cdot$ it can be shown that infinite reduction is not possible for any admissible arguments given. □

Lemma 4.2.7. *All process equations in G_1^3 are in pre-EGNF.*

Proof. It is easy to see that S_2 produces terms that satisfy the syntax (4.4) from Definition 4.1.6. The transformation S_1 can add only $+$, $\sum$ or $\lhd\rhd$ operations to them at the correct places, with regard to the syntax (4.4). The only interesting transformation to consider is $S_1\left(S, \sum_{d:D} p\right) \to \sum_{d:D} S_1(S \,\&\, d{:}D, p)$, as we need to show that p is not of the form $p_1 + p_2$. This follows from the fact that p satisfies the syntax defined in Table 4.1. □

Proposition 4.2.8. *For any process name X in G_1^2 we have $(\mathsf{X}, G_1^3) = (\mathsf{X}, G_1^2)$.*

Proof. The statement follows from Lemma 3.5.2. □

The transformation described in this subsection does not increase the size of terms. The number of process equations may increase linearly in the size of terms in the original system.

4.2.4 Guarding

Next we transform the equations of G_1^3 in such a way that each sequential term starts with an action (or τ). To this end, we define the function $guard : DVar \times Terms(|G|) \to Terms(|G|)$ in the following way:

$$guard\Big(S, \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i \lhd c_i \rhd \delta\Big) = rewr\Big(\sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} guard(S \cup \{\overrightarrow{e_i}\}, p_i) \lhd c_i \rhd \delta\Big)$$
$$guard(S, \mathsf{a}(\overrightarrow{t})) = \mathsf{a}(\overrightarrow{t})$$
$$guard(S, \mathsf{Y}(\overrightarrow{t})) = guard\Big(S, S_0\big(S \setminus \{pars(\mathsf{Y})\}, rhs(\mathsf{Y})\big)\big[pars(\mathsf{Y}) := \overrightarrow{t}\big]\Big)$$
$$guard(S, p_1 \cdot p_2) = rewr'(guard(S, p_1) \cdot p_2)$$

Here we use functions $rewr$ and S_0 from previous subsections. The function $rewr'$ represents the rewrite system of $rewr$ extended with the following rule (which is a directed version of the axiom (A4)).

$$(x + y) \cdot z \to x \cdot z + y \cdot z \tag{RA4}$$

It is clear that termination of *rewr'* can be proven similarly to Proposition 4.2.2. The function *guard* keeps track of the free variables that can occur in a term that is being guarded. In case we do the replacement of a process name by the right-hand side of its defining equation (third clause), we first rename its bound variables so that they do not become bound twice, then we substitute the values of the parameters, and then apply *guard* to the resulting term.

Proposition 4.2.9. *For any finite system G_1^3 with acyclic PNUDG, and any process name* X *in it, the function guard is well-defined on* $rhs(\mathsf{X}, G_1^3)$.

Proof. Let n be the number of equations in G_1^3. The only clause that makes the argument of *guard* larger is the third one. Due to the fact that PNUDG is acyclic, this rule cannot be applied more than n times deep (otherwise for some process name Z we would have a cycle). □

We define the system G_1^4 in the following way. For each equation

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta$$

in G_1^3 we put

$$\mathsf{X}(\overrightarrow{d{:}D}) = guard\Big(\{\vec{d}\}, \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta\Big)$$

into G_1^4.

Lemma 4.2.10. *The equations in G_1^4 are in pre-EGNF and all sequential process terms in the right-hand sides of its equations start with an action.*

Proof. Due to Proposition 4.2.9 we can apply induction on the definition of *guard*. The second and third clauses of the definition are trivial. In the first clause the only rules in Tables 4.2 that can be applied are (RCOND7), (RSUM12), (RCOND4) and (RSUM4), which bring the right-hand side to the desired form. (In case the inner *guard* returns δ, the rewrite rules that can be applied are (RCOND0′), (RSUM1′) and (RA6).) The fourth clause is brought to the desired form by applying (RA4), and then (RSUM5) and (RCOND6) from Table 4.2. (in case the inner *guard* returns δ, the rewrite rule that can be applied is (RA7).) □

Proposition 4.2.11. *Let G_1^3 and G_1^4 be defined as above. Then $G_1^3 = G_1^4$.*

Proof. According to Lemma 3.5.3 and Lemma 3.5.1 all transformations performed by *guard* lead to equivalent systems. We note that care has been taken to rename some data variables during the substitution (in the third clause of *guard* definition) in order to make the substitution and the following applications of the axioms sound. □

The transformation performed in this step does not increase the number of equations, but their sizes may grow exponentially, due to application of (RA4). An example of such an exponential growth is given below.

Example 4.2.12. Let n be a natural number and let the system of process equations G contain the following n equations.

$$\mathsf{X}_0 = \mathsf{a} + \mathsf{b}$$
$$\ldots$$
$$\mathsf{X}_n = \mathsf{X}_{n-1} \cdot \mathsf{a} + \mathsf{X}_{n-1} \cdot \mathsf{b}$$

By induction on n it is easy to show that after applying guarding we get $\mathsf{X}_n = \sum_{p \in \{\mathsf{a},\mathsf{b}\}^{n+1}} p$ where $\{\mathsf{a},\mathsf{b}\}^n$ is the set of all strings of length n consisting of a and b occurrences. Indeed, for $n = 0$ this is trivial. For $n > 0$ we get

$$\mathsf{X}_n \approx \left(\sum_{p \in \{\mathsf{a},\mathsf{b}\}^n} p \right) \cdot \mathsf{a} + \left(\sum_{p \in \{\mathsf{a},\mathsf{b}\}^n} p \right) \cdot \mathsf{b} \overset{(A4)}{\approx} \sum_{p \in \{\mathsf{a},\mathsf{b}\}^n} (p \cdot \mathsf{a}) + \sum_{p \in \{\mathsf{a},\mathsf{b}\}^n} (p \cdot \mathsf{b}) \approx \sum_{p \in \{\mathsf{a},\mathsf{b}\}^{n+1}} p$$

This example shows that the term in the right-hand side of the equation for X_n contains 2^n summands after the transformation.

4.2.5 Postprocessing

Finally, we transform all equations of G_1^4 into EGNF. This transformation can be seen as a simple postprocessing step in which we eliminate all actions that appear not leftmost in the right-hand sides in the equations. This elimination is obtained by introducing a new process name X_a for each action a that occurs inside the process terms p_i, with parameters corresponding to those of the action. Thus we add equations $\mathsf{X}_\mathsf{a}(\overrightarrow{d_\mathsf{a}{:}D_\mathsf{a}}) = \mathsf{a}(\overrightarrow{d_\mathsf{a}})$ to the system, and replace the occurrences of the action $\mathsf{a}(\overrightarrow{t})$ by $\mathsf{X}_\mathsf{a}(\overrightarrow{t})$.

Proposition 4.2.13. *Let the system G_1' of process equations be obtained after the postprocessing of the system G_1^4 as described above. Then for all $\mathsf{X} \in |G_1^4|$ we have $(\mathsf{X}, G_1') = (\mathsf{X}, G_1^4)$ and G_1' is in EGNF.*

Proof. According to Lemma 3.5.2 this transformation is correct and leads to a system that obviously is in EGNF. □

As a possible optimization during the postprocessing step, the following slightly different strategy can be applied. If we encounter a subterm $\mathsf{a} \cdot \mathsf{Y}$ in p_i, we replace it by a new process name (with the parameters for both a and Y), and add the equation for it to the system. This optimization goes along the lines of a so-called *regular linearization procedure* (see Conclusion), which is a more general case of such an optimization.

Summary. In this section we described the transformation of a finite system $G = G_1 \cup G_2$ with acyclic PNUDG and G_2 containing all parallel pCRL process equations into a system $G' = G'_1 \cup G_2$ with G'_1 in EGNF. For each $\mathsf{X} \in |G_1|$,

$$\begin{array}{rcll}(\mathsf{X}, G_1) & = & (\mathsf{X}, G_1^1) & (\text{“Preprocessing”, by Proposition 4.2.1}) \\ & = & (\mathsf{X}, G_1^2) & (\text{“Rewriting”, by Proposition 4.2.4}) \\ & = & (\mathsf{X}, G_1^3) & (\text{“Adding new equations”, by Proposition 4.2.8}) \\ & = & (\mathsf{X}, G_1^4) & (\text{“Guarding”, by Proposition 4.2.11}) \\ & = & (\mathsf{X}, G'_1) & (\text{“Postprocessing”, by Proposition 4.2.13}).\end{array}$$

By Lemma 3.3.5 it follows that $(\mathsf{X}, G) = (\mathsf{X}, G')$ for each $\mathsf{X} \in |G|$.

4.3 Collapsing into One Equation

In this section we transform the system of process equations $G' = G'_1 \cup G_2$ where G'_1 is in EGNF (cf. Definition 4.1.7) into $G'' = G''_1 \cup G'_2$, where

- G''_1 consists of a single process equation with a specially constructed parameter list;
- if G_2 is not empty, it is transformed into G'_2 with the same set $|G_2|$ of process names, but taking the effect of the transformation from G'_1 into G''_1 into account (references to G'_1 process identifiers may have to be adapted).

4.3.1 Formal Parameters Harmonization

In this subsection we make the formal parameters of all (non-parallel) pCRL process names in G'_1 uniform, and adapt the parallel pCRL equations in G_2 in an appropriate way. This is done to be able to compress all (non-parallel) pCRL equations into one. The harmonization is defined by the following steps.

1. We rename the data variables with the same names but with different types in different processes. This can easily be done (see Section 4.2.1).
2. We create the common list of data parameters $\overrightarrow{d{:}D}$ by taking the *set* of all data parameters in the pCRL equations, and giving some order to it.
3. For each pCRL process name X in G'_1 we define a mapping M_{X} from its parameter list $\overrightarrow{D_{\mathsf{X}}}$ to the common parameter list $\overrightarrow{D}$. This mapping is such that each newly created parameter is a constant. (Recall that a correct μCRL specification contains constants for each declared data sort.)
4. Then we replace all left-hand sides of the pCRL process equations $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}})$ by $\mathsf{X}(\overrightarrow{d{:}D})$, and *all* pCRL process name occurrences $\mathsf{Y}(\overrightarrow{t})$ in the right-hand sides of *all* the equations in G' by $\mathsf{Y}(M_{\mathsf{Y}}(\overrightarrow{t}))$.

We demonstrate this step by an example.

Example 4.3.1. Let $G = \{\mathsf{X}(n{:}Nat) = \mathsf{a}(n)\cdot\mathsf{X}(succ(n)), \mathsf{Y}(b{:}Bool) = \mathsf{b}(b)\cdot\mathsf{Y}(\neg b), \mathsf{Z} = \mathsf{X}(0) \parallel \mathsf{Y}(\mathbf{t})\}$ be a given system of process equations, where Z is the only parallel pCRL equation, and we are interested in the process definition (Z, G). After applying parameter harmonization we get the following system of equations: $G' = \{\mathsf{X}(n{:}Nat, b{:}Bool) = \mathsf{a}(n)\cdot\mathsf{X}(succ(n), b), \mathsf{Y}(n{:}Nat, b{:}Bool) = \mathsf{b}(b)\cdot\mathsf{Y}(n, \neg b), \mathsf{Z} = \mathsf{X}(0, \mathbf{f}) \parallel \mathsf{Y}(0, \mathbf{t})\}$.

Proposition 4.3.2. *Let the system $G_1^5 \cup G_2^1$ of process equations be obtained after harmonization of the system $G_1' \cup G_2$ as described above. Then for all $\mathsf{X} \in |G_1'|$ we have $(\mathsf{X}(M_{\mathsf{X}}(\overrightarrow{d_{\mathsf{X}}})), G_1^5) = (\mathsf{X}(\overrightarrow{d_{\mathsf{X}}}), G_1')$, and for all $\mathsf{X} \in |G_2|$, $(\mathsf{X}, G_1^5 \cup G_2^1) = (\mathsf{X}, G_1' \cup G_2)$.*

Proof. By Lemma 3.5.4 it follows that this transformation yields an equivalent system of equations. □

We remark that a more optimal strategy in terms of the number of data parameters, than 'global harmonization', is to merge as many parameters as possible. This can be achieved by renaming parameters of some processes so that they match the parameters of other processes, and therefore are not introduced in the general parameter list. In this case the number of parameters of some type s in the general list will be the maximal number of parameters of this type in an equation. A drawback of this optimization is the fact that we may lose parameter name information for some process names.

4.3.2 Making One Process Equation

In this subsection we combine n process equations from G_1^5 with the same formal parameters into one equation. This is done by adding a data parameter $s{:}StateN$ that represents the process names from $|G_1^5|$ to the parameters; adding a condition to each summand of each equation which checks that the value of data parameter s is the appropriate one; and combining all right-hand sides into one alternative composition. The data type *StateN* is an enumerated data type with equality predicate. Natural numbers could be used for *StateN*. A finite data type is sufficient though.

More precisely, let G_1^5 be a system of n μCRL process equations in EGNF with the same formal parameters.

$$
\begin{aligned}
\mathsf{X}^1(\overrightarrow{d{:}D}) &= \sum_{i\in I^1} \sum_{\overrightarrow{e_i{:}E_i^1}} \mathsf{a}_i^1(\overrightarrow{f_i^1}(\overrightarrow{d, e_i})) \cdot p_i^1(\overrightarrow{d, e_i}) \lhd c_i^1(\overrightarrow{d, e_i}) \rhd \delta \\
&+ \sum_{j\in J^1} \sum_{\overrightarrow{e_j{:}E_j^1}} \mathsf{a}_j^1(\overrightarrow{f_j^1}(\overrightarrow{d, e_j})) \lhd c_j^1(\overrightarrow{d, e_j}) \rhd \delta \\
&\vdots \\
\mathsf{X}^n(\overrightarrow{d{:}D}) &= \sum_{i\in I^n} \sum_{\overrightarrow{e_i{:}E_i^n}} \mathsf{a}_i^n(\overrightarrow{f_i^n}(\overrightarrow{d, e_i})) \cdot p_i^n(\overrightarrow{d, e_i}) \lhd c_i^n(\overrightarrow{d, e_i}) \rhd \delta \\
&+ \sum_{j\in J^n} \sum_{\overrightarrow{e_j{:}E_j^n}} \mathsf{a}_j^n(\overrightarrow{f_j^n}(\overrightarrow{d, e_j})) \lhd c_j^n(\overrightarrow{d, e_j}) \rhd \delta
\end{aligned}
$$

We define the system G_1^6 as a single EGNF process equation in the following way:

$$
\begin{aligned}
&\mathsf{X}(s{:}\mathit{StateN}, \overrightarrow{d{:}D})\\
&\quad = \sum_{i\in I^1}\sum_{\overrightarrow{e_i:E_i^1}} \mathsf{a}_i^1(\overrightarrow{f_i^1}(\overrightarrow{d,e_i}))\cdot S(p_i^1(\overrightarrow{d,e_i})) \lhd c_i^1(\overrightarrow{d,e_i}) \wedge s=1 \rhd \delta\\
&\qquad + \sum_{j\in J^1}\sum_{\overrightarrow{e_j:E_j^1}} \mathsf{a}_j^1(\overrightarrow{f_j^1}(\overrightarrow{d,e_j})) \lhd c_j^1(\overrightarrow{d,e_j}) \wedge s=1 \rhd \delta\\
&\quad\ \vdots\\
&\qquad + \sum_{i\in I^n}\sum_{\overrightarrow{e_i:E_i^n}} \mathsf{a}_i^n(\overrightarrow{f_i^n}(\overrightarrow{d,e_i}))\cdot S(p_i^n(\overrightarrow{d,e_i})) \lhd c_i^n(\overrightarrow{d,e_i}) \wedge s=n \rhd \delta\\
&\qquad + \sum_{j\in J^n}\sum_{\overrightarrow{e_j:E_j^n}} \mathsf{a}_j^n(\overrightarrow{f_j^n}(\overrightarrow{d,e_j})) \lhd c_j^n(\overrightarrow{d,e_j}) \wedge s=n \rhd \delta
\end{aligned}
$$

where $S(\mathsf{X}^s(\overrightarrow{t})) = \mathsf{X}(s, \overrightarrow{t})$ and S distributes over all process operations.

During the current step we construct the system G_1^6 consisting of the single equation for X and the set G_2^2 being G_2^1 with all pCRL process terms $\mathsf{X}^i(\overrightarrow{t})$ replaced by $\mathsf{X}(i, \overrightarrow{t})$ for each $1 \leq i \leq n$.

Proposition 4.3.3. *Let G_1^5 be a system of n process equations in EGNF, each with formal parameters $\overrightarrow{d{:}D}$, and let StateN enumerate $1, \ldots, n$. Let furthermore $G_1^5 \cup G_2^1$ be a system of parallel pCRL process equations and $G_1^6 \cup G_2^2$ be the result of the transformation described above. Then for any $s{:}\mathit{StateN}$, data term vector $\overrightarrow{t}$, and any $\mathsf{X}^s \in |G_1^5|$, $(\mathsf{X}(s, \overrightarrow{t}), G_1^6) \overset{cond}{=} (\mathsf{X}^s(\overrightarrow{t}), G_1^5)$. Finally, for each $\mathsf{X} \in |G_2^1|$, $(\mathsf{X}, G_1^6 \cup G_2^2) \overset{cond}{=} (\mathsf{X}, G_1^5 \cup G_2^1)$.*

Proof. The equivalence is easy to derive with the following functions: $w_{\mathsf{X}^i}(\overrightarrow{t}) = \mathsf{X}(i, \overrightarrow{t})$ for each $i{:}\mathit{StateN}$, and $w_{\mathsf{X}}(s, \overrightarrow{t}) = \mathsf{X}^s(\overrightarrow{t})$. Note that identities of sort *StateN* are used in the derivations. □

Example 4.3.4. Let G' be as defined in Example 4.3.1. We collapse the equations for process names X and Y into one, and get the following system (in this case we can use booleans to represent the sort *StateN*):

$$
\begin{aligned}
G'' = \{\mathsf{T}(s{:}\mathit{Bool}, n{:}\mathit{Nat}, b{:}\mathit{Bool}) &= \mathsf{a}(n)\cdot \mathsf{T}(s, \mathit{succ}(n), b) \lhd \mathit{eq}(s, \mathbf{f}) \rhd \delta\\
&\quad + \mathsf{b}(b)\cdot \mathsf{T}(s, n, \neg b) \lhd \mathit{eq}(s, \mathbf{t}) \rhd \delta,\\
\mathsf{Z} = \mathsf{T}(\mathbf{f}, 0, \mathbf{f}) \parallel \mathsf{T}(\mathbf{t}, 0, \mathbf{t})\}.&
\end{aligned}
$$

4.4 Introduction of a Stack

The final step in the linearization of pCRL processes consists of the introduction of a stack parameter which allows to model a sequential composition of process names with parameters as a single process term. In the case that such sequential compositions do not occur in the equation, we do not apply this step. For the particular transformation described here, it is necessary that the process equation to be transformed is data-parametric. This need not be the case after application of all preceding transformation steps. For instance the equation $\mathsf{X} = \mathsf{a}\cdot\mathsf{X}\cdot\ldots\cdot\mathsf{X}+\mathsf{b}$ does not have a data parameter. In this case we need to add a dummy data parameter (over a singleton data type, cf. Lemma 3.5.4) to apply the following transformation.

Let G_1^6 contain a single pCRL process equation in EGNF:

$$\begin{aligned}\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot\mathsf{X}(\overrightarrow{t_i^1})\cdot\ldots\cdot\mathsf{X}(\overrightarrow{t_i^{n_i}}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta + \\ &\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta\end{aligned}$$

We define G_1'' by the single process equation for Z in the following way:

$$\begin{aligned}&\mathsf{Z}(st{:}Stack) = \\ &\quad\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{get(st),e_i}))\cdot\mathsf{Z}(push(\overrightarrow{t_i^1},\ldots push(\overrightarrow{t_i^{n_i}},pop(st))\cdots)) \\ &\qquad\qquad\qquad\qquad\lhd st\neq\langle\rangle\wedge c_i(\overrightarrow{get(st),e_i})\rhd\delta \\ &+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{get(st),e_j}))\cdot\mathsf{Z}(pop(st)) \\ &\qquad\qquad\qquad\qquad\lhd st\neq\langle\rangle\wedge pop(st)\neq\langle\rangle\wedge c_j(\overrightarrow{get(st),e_j})\rhd\delta \\ &+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{get(st),e_j}))\lhd st\neq\langle\rangle\wedge pop(st)=\langle\rangle\wedge c_j(\overrightarrow{get(st),e_j})\rhd\delta\end{aligned}$$

where $\overrightarrow{get(st)} = get_1(st),\ldots,get_n(st)$.

The data type *Stack* is a standard stack data type with constructors $\langle\rangle$ representing the empty stack, and $push(\overrightarrow{t},st)$ inserting the new element $\overrightarrow{t}$ to the top of the stack st. We use the equality predicate on stacks, but a predicate that checks if a stack is empty can be used instead. The function $get_i(st)$ returns the ith element from the top of st, and the function $pop(st)$ returns the stack value st without its top element. See [59] for details on implementing data types in μCRL. To prove the following proposition we use an induction principle on the data type *Stack*, namely that every value of type stack is either empty or the result of an insertion to another value of this type.

During the current step we construct the system G_1'' consisting of the single equation for X and the set G_2' being G_2^2 with all pCRL process terms $\mathsf{X}(\overrightarrow{t})$ replaced by $\mathsf{Z}(push(\overrightarrow{t},\langle\rangle))$.

Proposition 4.4.1. *Let systems G_1^6 and G_1'' as described above be given. Then for any data term vector $\overrightarrow{t}$ we have $(\mathsf{X}(\overrightarrow{t}),G_1^6)\overset{ind}{\Rightarrow}(\mathsf{Z}(push(\overrightarrow{t},\langle\rangle)),G_1'')$. Let furthermore $G_1^6\cup G_2^2$ be a system of parallel pCRL process equations and $G_1''\cup G_2'$ be the result of the transformation described above. Then for any $\mathsf{X}\in|G_2^2|$, $(\mathsf{X},G_1^6\cup G_2^2)\overset{ind}{\Rightarrow}(\mathsf{X},G_1''\cup G_2')$.*

Proof. We define $w_{\mathsf{Z}}(st)$ for all well-defined closed terms of sort *Stack* in the following way:

$$
\begin{aligned}
&w_{\mathsf{Z}}(\langle\rangle) = \delta\\
&w_{\mathsf{Z}}(push(\overrightarrow{t}, \langle\rangle)) = \mathsf{X}(\overrightarrow{t})\\
&w_{\mathsf{Z}}(push(\overrightarrow{t}, push(\overrightarrow{t'}, st'))) = \mathsf{X}(\overrightarrow{t}) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st')).
\end{aligned}
$$

It is clear from this definition that

$$
\begin{aligned}
&w_{\mathsf{Z}}(push(\overrightarrow{t^1}, \ldots push(\overrightarrow{t^n}, \langle\rangle)\cdots)) \approx \mathsf{X}(\overrightarrow{t^1}) \cdot \ldots \cdot \mathsf{X}(\overrightarrow{t^n})\\
&w_{\mathsf{Z}}(push(\overrightarrow{t^1}, \ldots push(\overrightarrow{t^n}, push(\overrightarrow{t'}, st')))\cdots)) \approx \mathsf{X}(\overrightarrow{t^1}) \cdot \ldots \cdot \mathsf{X}(\overrightarrow{t^n}) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st')).
\end{aligned}
$$

To prove the implication we need to derive two proof obligations. The first one is

$$
\mathsf{X}(\overrightarrow{t}) \approx w_{\mathsf{Z}}(push(\overrightarrow{t}, \langle\rangle))
$$

which clearly follows from the definition of w_{Z}. The second proof obligation is the equation for Z with $\mathsf{Z}(st)$ replaced by $w_{\mathsf{Z}}(st)$. We prove it for all parameters of Z that are well-defined closed terms of sort *Stack*. We have three cases:

1. For the parameter value $\langle\rangle$ both sides of the equation are equal to δ.

2. For the parameter value $push(\overrightarrow{t}, \langle\rangle)$ the left-hand side of the equation for Z becomes $\mathsf{X}(\overrightarrow{t})$, and the right-hand side is exactly the right-hand side of the equation for X because

$$
\begin{aligned}
&get(push(\overrightarrow{t}, \langle\rangle)) \approx \overrightarrow{t}\\
&pop(push(\overrightarrow{t}, \langle\rangle)) \approx \langle\rangle\\
&push(\overrightarrow{t}, \langle\rangle) \neq \langle\rangle \approx \mathbf{t}
\end{aligned}
$$

   This identity is trivially derivable from the equation for X.

3. For the parameter value $push(\overrightarrow{t}, push(\overrightarrow{t'}, st'))$ the left-hand side is equal to $\mathsf{X}(\overrightarrow{t}) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st'))$ so we need to show that the right-hand side is also equal to this term. We use the following identities on *Stack* in this case:

$$
\begin{aligned}
get(push(\overrightarrow{t}, push(\overrightarrow{t'}, st'))) &\approx \overrightarrow{t}\\
pop(push(\overrightarrow{t}, push(\overrightarrow{t'}, st'))) &\approx push(\overrightarrow{t'}, st')\\
pop(push(\overrightarrow{t}, push(\overrightarrow{t'}, st'))) \neq \langle\rangle &\approx \mathbf{t}\\
push(\overrightarrow{t}, push(\overrightarrow{t'}, st')) \neq \langle\rangle &\approx \mathbf{t}
\end{aligned}
$$

   When we apply w_{Z} to the equation for Z and use the identities of the sort *Stack*, we get the following identity:

$$
\begin{aligned}
&\mathsf{X}(\overrightarrow{t}) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st')) \\
&\quad \approx \sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{t, e_i})) \cdot \mathsf{X}(\overrightarrow{t_i^1}) \cdot \ldots \cdot \mathsf{X}(\overrightarrow{t_i^{n_i}}) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st')) \lhd c_i(\overrightarrow{t, e_i}) \rhd \delta \\
&\quad + \sum_{j \in J} \sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{t, e_j})) \cdot w_{\mathsf{Z}}(push(\overrightarrow{t'}, st')) \lhd c_j(\overrightarrow{t, e_j}) \rhd \delta
\end{aligned}
$$

This identity is derivable from the equation for X by applying the axioms (Cond6), (SUM5) and (A4).

□

The following example [79] shows that the reverse implication does not hold in every model. It is easy to see that if data parameters do not matter, the stack is isomorphic to a counter which can be implemented by means of natural numbers.

Example 4.4.2. Let $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X} \cdot \mathsf{X}\}$ and $G_2 = \{\mathsf{Z}(n{:}Nat) = \mathsf{a} \cdot \mathsf{Z}(succ(n))\}$. Consider the model with integers $\mathbb{Z}$ as the carrier set, and the operations $\cdot \to +, \mathsf{a} \to -1$. The equation in G_1 has the unique solution $\mathsf{X} = 1$, while the equation in G_2 has infinitely many solutions $\mathsf{Z}(n) = n + c$, where $c \in \mathbb{Z}$. For a more elaborated model that includes interpretations of other μCRL operations see Example 4.5.2.

Summary. The previous section and this one consider the transformation of a finite system $G' = G_1' \cup G_2$, where G_1' in EGNF, into a system $G'' = G_1'' \cup G_2'$ with G_1'' an LPE and G_2' appropriately updated. For each $\mathsf{X} \in |G'|$,

$$
\begin{array}{lcll}
(\mathsf{X}, G') & = & (\mathsf{X}', G_1^5 \cup G_2^1) & (\text{“Harmonization”, by Proposition 4.3.2}) \\
 & \overset{cond}{=} & (\mathsf{X}'', G_1^6 \cup G_2^2) & (\text{“One equation”, by Proposition 4.3.3}) \\
 & \overset{ind}{\Rightarrow} & (\mathsf{X}''', G'') & (\text{“One LPE”, by Proposition 4.4.1}).
\end{array}
$$

Here the primed versions of X represent the possible updates of parameters, as prescribed by the propositions mentioned.

4.5 From Parallel pCRL to LPE

As the result of the previous section we have obtained $G'' = G_1'' \cup G_2'$, where G_1'' is an LPE and G_2' a (possibly empty) set of parallel pCRL process equations. In this section we show that the parallel part of G'' can be eliminated. First we take a general point of view, and show that LPEs are closed under the parallel pCRL process operations, viz. parallel composition, encapsulation, hiding, and renaming (see Definition 4.1.2). Then we show that with these results and those from Sections 4.2 and 4.3, the transformation of G'' into a single LPE can be carried out. We note that the transformation described in this section is uni-directional, and we give counterexamples for the associated reverse implications.

4.5.1 Parallel Composition of LPEs

Let G be a system of process equations in which each of $(\mathsf{X}(\overrightarrow{d_\mathsf{X}}), G)$ and $(\mathsf{Y}(\overrightarrow{d_\mathsf{Y}}), G)$ is defined by an LPE, and that contains an equation $\mathsf{Z}(\overrightarrow{d_\mathsf{X}, d_\mathsf{Y}}) = \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \parallel \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Assume that the LPEs for X and Y have no common data variables, and are defined in the following way:

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) =& \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i}))\cdot \mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta\\
&+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta\\
\mathsf{Y}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}) =& \sum_{i\in I'}\sum_{\overrightarrow{e'_i:E'_i}} \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d_\mathsf{Y}, e'_i}))\cdot \mathsf{Y}(\overrightarrow{g'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \lhd c'_i(\overrightarrow{d_\mathsf{Y}, e'_i}) \rhd \delta\\
&+ \sum_{j\in J'}\sum_{\overrightarrow{e'_j:E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d_\mathsf{Y}, e'_j})) \lhd c'_j(\overrightarrow{d_\mathsf{Y}, e'_j}) \rhd \delta
\end{aligned}$$

where $I \cap J = I' \cap J' = \emptyset$. We construct the equation for $\mathsf{Z}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}, d_\mathsf{Y}{:}D_\mathsf{Y}})$, being equal to $\mathsf{X}(\overrightarrow{d_\mathsf{X}}) \parallel \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$, as follows.

$$\begin{aligned}
&\mathsf{Z}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}, d_\mathsf{Y}{:}D_\mathsf{Y}}) =\\
&\quad \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i}))\cdot \mathsf{Z}(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i}), \overrightarrow{d_\mathsf{Y}}) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta\\
&\quad + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j}))\cdot \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta\\
&\quad + \sum_{i\in I'}\sum_{\overrightarrow{e'_i:E'_i}} \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d_\mathsf{Y}, e'_i}))\cdot \mathsf{Z}(\overrightarrow{d_\mathsf{X}}, \overrightarrow{g'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \lhd c'_i(\overrightarrow{d_\mathsf{Y}, e'_i}) \rhd \delta\\
&\quad + \sum_{j\in J'}\sum_{\overrightarrow{e'_j:E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d_\mathsf{Y}, e'_j}))\cdot \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \lhd c'_j(\overrightarrow{d_\mathsf{Y}, e'_j}) \rhd \delta\\
&\quad + \sum_{(k,l)\in I\gamma I'}\ \sum_{\overrightarrow{e_k:E_k, e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}))\cdot \mathsf{Z}(\overrightarrow{g_k}(\overrightarrow{d_\mathsf{X}, e_k}), \overrightarrow{g'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_k}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta\\
&\quad + \sum_{(k,l)\in I\gamma J'}\ \sum_{\overrightarrow{e_k:E_k, e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}))\cdot \mathsf{X}(\overrightarrow{g_k}(\overrightarrow{d_\mathsf{X}, e_k}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_k}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta\\
&\quad + \sum_{(k,l)\in J\gamma I'}\ \sum_{\overrightarrow{e_k:E_k, e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}))\cdot \mathsf{Y}(\overrightarrow{g'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_k}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta\\
&\quad + \sum_{(k,l)\in J\gamma J'}\ \sum_{\overrightarrow{e_k:E_k, e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_k}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta
\end{aligned}$$

where $P\gamma Q = \{(p,q) \in P \times Q \mid \gamma(\mathsf{a}_p, \mathsf{a}'_q) \text{ is defined}\}$.

Proposition 4.5.1. *Let G' contain the equations for X, Y and Z defined above. Let G contain the equations for X and Y, and the equation $\mathsf{Z}(\overrightarrow{d_\mathsf{X}, d_\mathsf{Y}}) = \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \parallel \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Then $(\mathsf{Z}, G) \Rightarrow (\mathsf{Z}, G')$.*

Proof. We use the identity mapping for $w_\mathsf{X}, w_\mathsf{Y}, w_\mathsf{Z}$. Then the equations for X and Y are proven trivially because they are the same in G and G'. To prove the equation for Z first apply the axiom (CM1) to get $\mathsf{Z} = (\mathsf{X}(\overrightarrow{d_\mathsf{X}}) \mathbin{⌊\!⌊} \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) + \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \mathbin{⌊\!⌊} \mathsf{X}(\overrightarrow{d_\mathsf{X}})) + \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \mid \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Then we replace X and Y in the left-hand sides of $⌊\!⌊$ and in both sides of $\mid$ by their right-hand sides. After that we apply the axioms (CM4), (SUM6), (Cond8), (CM2) and (CM3) to eliminate $⌊\!⌊$, and the axioms (CM8), (CM9), (SUM7), (SUM7′), (Cond9), (Cond9′), (CM5), (CM6), (CM7), (CF1), (CF2),(CT1), (CT2), (CD1), (CD2) to eliminate $\mid$. Note that before applying the axioms for sums we might need to apply (SUM2), and after elimination $⌊\!⌊$ and $\mid$ we might need to apply (A7) and (A6). After that we apply the identity $x \parallel y \approx y \parallel x$, which is derivable from the axioms (cf. Lemma 2.2.6), to replace all occurrences of $\mathsf{Y}(\overrightarrow{t'}) \parallel \mathsf{X}(\overrightarrow{t})$ by $\mathsf{X}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{t'})$, and finally we replace all $\mathsf{X}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{t'})$ by $\mathsf{Z}(\overrightarrow{t}, \overrightarrow{t'})$ using the equation for Z in G. As the result we get the equation for Z in G'. □

In the following example we present a model of μCRL based on the trace model [44], but in which the sequential composition operation is commutative and idempotent. This model is used in Example 4.5.3 to show that the reverse implication of Proposition 4.5.1 does not hold in every model.

Example 4.5.2. Let *ActLab* be a finite set of action labels and γ be the totally undefined function. Consider the model with carrier set $\left(2^{\left(2^{ActLab}\setminus\{\emptyset\}\right)} \setminus \{\emptyset\}\right) \cup \{\top, \bot\}$, and the operations defined as follows:

- For each $\mathsf{a} \in \mathit{ActLab}$ $\mathsf{a}(\overrightarrow{t}) \to \{\{\mathsf{a}\}\}$
- $\delta \to \top$ and $\tau \to \bot$
- $+ \to \cup$, where $S \cup \top = \top \cup S = S$ and $S \cup \bot = \bot \cup S = \bot$
- $\cdot, \parallel, ⌊\!⌊, \mid \to *$, where $S * S' = \{s \cup s' \mid s \in S \wedge s' \in S'\}$, $S * \top = \top * S = \top$ and $S * \bot = \bot * S = S$.
- $\partial_H \to e_H$, where $e_H(\{\{\mathsf{a}\}\}) = \{\{\mathsf{a}\}\}$ if $\mathsf{a} \notin H$, $e_H(\{\{\mathsf{a}\}\}) = \top$ if $\mathsf{a} \in H$, $e_H(S \cup S') = e_H(S) \cup e_H(S')$, $e_H(S * S') = e_H(S) * e_H(S')$, $e_H(\top) = \top$, $e_H(\bot) = \bot$
- $\tau_I \to h_I$, where h_I is defined in a similar way as e_H.
- $\sum_{d:D} \to \mathit{id}$, where id is the identity mapping.
- $x \lhd c \rhd y \to \mathit{if}(c, x, y)$, where $\mathit{if}(c, x, y)$ is the if-then-else mapping.

Example 4.5.3. Let $G = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}, \mathsf{Y} = \mathsf{b} \cdot \mathsf{Y}, \mathsf{Z} = \mathsf{X} \parallel \mathsf{Y}\}$ and $G' = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}, \mathsf{Y} = \mathsf{b} \cdot \mathsf{Y}, \mathsf{Z} = \mathsf{a} \cdot \mathsf{Z} + \mathsf{b} \cdot \mathsf{Z}\}$. In the model defined in Example 4.5.2 the equations for X in both G and G' have the following solutions:

$$\{\{\mathsf{a}\}\}, \quad \{\{\mathsf{a},\mathsf{b}\}\}, \quad \{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}, \quad \top$$

while the equations for Y have the following solutions:

$$\{\{\mathsf{b}\}\}, \quad \{\{\mathsf{a},\mathsf{b}\}\}, \quad \{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}, \quad \top$$

The equation for Z in G has two solutions $\{\{\mathsf{a},\mathsf{b}\}\}$ and $\top$, while the equation for Z in G' has five solutions $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

4.5.2 Encapsulation, Hiding and Renaming of LPEs

Let G be an LPE defining X as in the previous section, A be a set of action labels, and R be a renaming function. We construct LPEs for Z_1 being equal to $\partial_A(\mathsf{X})$, Z_2 being equal to $\tau_A(\mathsf{X})$, and Z_3 being equal to $\rho_R(\mathsf{X})$, in the following way:

$$\begin{aligned}
\mathsf{Z}_1(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = &\sum_{i\in I_1}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\vec{f_i}(\overrightarrow{d_\mathsf{X}, e_i})) \cdot \mathsf{Z}_1(\vec{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
&+ \sum_{j\in J_1}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\vec{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta
\end{aligned}$$

Here and in the equations below we assume that $I_1 = \{i \in I \mid \mathsf{a}_i \notin A\}$ and $J_1 = \{j \in J \mid \mathsf{a}_j \notin A\}$.

$$\begin{aligned}
\mathsf{Z}_2(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = &\sum_{i\in I_1}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\vec{f_i}(\overrightarrow{d_\mathsf{X}, e_i})) \cdot \mathsf{Z}_2(\vec{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
&+ \sum_{j\in J_1}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\vec{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta \\
&+ \sum_{i\in I\setminus I_1}\sum_{\overrightarrow{e_i:E_i}} \tau \cdot \mathsf{Z}_2(\vec{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
&+ \sum_{j\in J\setminus J_1}\sum_{\overrightarrow{e_j:E_j}} \tau \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta \\
\mathsf{Z}_3(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} R(\mathsf{a}_i)(\vec{f_i}(\overrightarrow{d_\mathsf{X}, e_i})) \cdot \mathsf{Z}_3(\vec{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
&+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} R(\mathsf{a}_j)(\vec{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta
\end{aligned}$$

Proposition 4.5.4. *Let G'_1 contain the equations for X and Z_1 defined above, G'_2 contain the equations for X and Z_2 defined above, and G'_3 contain the equations for X and Z_3 defined above. Let G_1 contain the equations for X and $\mathsf{Z}_1(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \partial_A(\mathsf{X}(\overrightarrow{d_\mathsf{X}}))$,*

G_2 contain the equations for X *and* $\mathsf{Z}_2(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_A(\mathsf{X}(\overrightarrow{d_\mathsf{X}}))$*, and G_3 contain the equations for* X *and* $\mathsf{Z}_3(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \rho_R(\mathsf{X}(\overrightarrow{d_\mathsf{X}}))$*. Then we have $G_1 \Rightarrow G_1'$, $G_2 \Rightarrow G_2'$ and $G_3 \Rightarrow G_3'$.*

Proof. To prove the implications we use the identity mappings for $w_\mathsf{X}, w_{\mathsf{Z}_1}, w_{\mathsf{Z}_2}$ and w_{Z_3}. The equations for X are proven trivially. For the other equations we substitute X by its right-hand side and apply the axioms (D3), (SUM8), (D5), (D4), (D1), (D2), (A7), (A6) to push ∂_A inside; the axioms (T3), (SUM9), (T5), (T4), (T1), (T2) to push τ_A inside; the axioms (R3), (SUM10), (R5), (R4), (R1), (RT), (RD) to push ρ_R inside. After that we use the equations for $\mathsf{Z}_1, \mathsf{Z}_2, \mathsf{Z}_3$ in G_1, G_2, G_3 respectively to eliminate the operators ∂_A, τ_A and ρ_R completely and arrive at equations for $\mathsf{Z}_1, \mathsf{Z}_2, \mathsf{Z}_3$ in G_1', G_2', G_3' respectively. □

The following examples show that the reverse implications of the latter proposition do not hold in every model.

Example 4.5.5. Let $G_1 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_1 = \partial_{\{\mathsf{b}\}}(\mathsf{X})\}$ and $G_1' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_1 = \mathsf{a}\cdot\mathsf{Z}_1\}$. Consider the model from Example 4.5.2. The equations for X in both G_1 and G_1' have the following solutions:

$$\{\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{a}\},\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\},\quad \top$$

The equation for Z_1 in G_1 has two solutions $\{\{\mathsf{a}\}\}$ and $\top$, while the equation for Z_1 in G_1' has four solutions $\{\{\mathsf{a}\}\}$, $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

Example 4.5.6. Let $G_2 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Z}_2 = \tau_{\{\mathsf{a}\}}(\mathsf{X})\}$ and $G_2' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Z}_2 = \tau\cdot\mathsf{Z}_2\}$. Consider the branching bisimulation model [44]. The equation for Z_2 in G_2 has the unique solution $\mathsf{Z}_2 = \tau\cdot\delta$, while the equation for Z_2 in G_2' has infinitely many solutions $\mathsf{Z}_2 = \tau\cdot p$, where p is any element of the model.

Example 4.5.7. Let $G_3 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_3 = \rho_R(\mathsf{X})\}$ and $G_3' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_3 = \mathsf{a}\cdot\mathsf{Z}_3\}$, where $R(\mathsf{a}) = R(\mathsf{b}) = \mathsf{a}$. Consider the model from Example 4.5.2. The equation for Z_3 in G_3 has two solutions $\{\{\mathsf{a}\}\}$ and $\top$, while the equation for Z_3 in G_3' has four solutions $\{\{\mathsf{a}\}\}$, $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

4.5.3 Towards an LPE

Let $G'' = G_1'' \cup G_2'$ be a system of process equations with G_1'' an LPE and G_2' containing parallel pCRL process equations. If G_2' is empty we are done. Otherwise, let (X, G'') be the process definition to be transformed. We substitute the right-hand sides for all parallel pCRL process names (other than X) in G_2' and obtain the set G_2'' with a single process equation for X, such that $(\mathsf{X}, G'') = (\mathsf{X}, G_1'' \cup G_2'')$. We finish the description of our transformation of G'' into a single LPE by describing how G_2'' can be integrated with G_1''. A general strategy is to apply an innermost/outermost reduction along the lines of Propositions 4.5.1 and 4.5.4, occasionally adding or replacing process equations.

We consider a typical case (but note that many variants are conceivable):

$$\begin{array}{rcl} G_1'' & = & \{\mathsf{Y}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}) = p_\mathsf{Y}\} \\ G_2' & = & \{\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{u})))\} \end{array}$$

and proceed in a stepwise manner. First we reduce the $\|$-occurrence, so transform G_2' into

$$G_2^3 = \{\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_I(\partial_H(\mathsf{Z}(\overrightarrow{t}, \overrightarrow{u}))),\ \mathsf{Z}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}, \overrightarrow{e_\mathsf{Y}{:}D_\mathsf{Y}}) = \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \parallel \mathsf{Y}(\overrightarrow{e_\mathsf{Y}})\}$$

where $\overrightarrow{e_\mathsf{Y}}$ is a fresh copy of $\overrightarrow{d_\mathsf{Y}}$. With Lemma 3.5.2 it follows that for all $\mathsf{Y} \in |G''|$, $(\mathsf{Y}, G'') = (\mathsf{Y}, G_1'' \cup G_2^3)$. According to Proposition 4.5.1, there exists a system H with Z defined by a number of *linear* equations in the process names Z and Y such that $(\mathsf{Z}, G_1'' \cup G_2^3) \stackrel{ind}{\Rightarrow} (\mathsf{Z}, H)$, and for the remaining process names $\mathsf{Y} \in |G''|$, $(\mathsf{Y}, G'') = (\mathsf{Y}, G_1'' \cup G_2^3) \stackrel{ind}{\Rightarrow} (\mathsf{Y}, H)$. Comparing the newly created system H of process equations with G'', we see that it contains one parallel pCRL operation less, and one more pCRL process equation consisting of the linear equation for Z. Next, with Propositions 4.3.2 and 4.3.3 this system can be transformed into a system H' that contains a single LPE, say over process name U, and the equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_I(\partial_H(\mathsf{U}(\overrightarrow{u})))$ where application of these propositions prescribes the value vector $\overrightarrow{u}$. With Proposition 4.5.4 we can resolve the encapsulation and hiding operation in a similar fashion. This yields a system of process equations H'' that consists of an LPE over process name V and the equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \mathsf{V}(\overrightarrow{v})$, and $(\mathsf{X}, H') \Rightarrow (\mathsf{X}, H'')$. Now the last step of this final transformation is the conclusion $(\mathsf{X}, H'') = (\mathsf{V}(\overrightarrow{v}), L)$, where L contains only the LPE for V.

The description above illustrates the last part of our transformation. Without further proof we state the following result.

Proposition 4.5.8. *Let $G'' = G_1'' \cup G_2'$ be a system of process equations as described above (G_1'' an LPE, and G_2' containing parallel pCRL process equations). Then G'' can be transformed via innermost/outermost reduction into a system L that contains one single LPE, and that satisfies $(\mathsf{X}, G'') \stackrel{ind}{\Rightarrow} (\mathsf{X}'(\overrightarrow{t_{\mathsf{X}'}}), L)$ for a certain value vector $\overrightarrow{t_{\mathsf{X}'}}$.*

Chapter 5

Linearization in μCRL

In this chapter we present an extension of the linearization algorithm to the setting of full μCRL. The main difference lies in the fact that we are dealing with recursive occurrences of parallel composition and renaming operations. This allows to express systems with dynamic creation of parallel components. The first part of the algorithm, up to obtaining of the single equation, is a rather straightforward extension of the pCRL algorithm presented in Chapter 4. The data type introduction and the data type in itself is significantly more involved than the stack introduction described in Section 4.4.

5.1 Transformation to Post-PEGNF

As input for the linearization procedure we take a μCRL process definition $(\mathsf{X}(\overrightarrow{t}), G)$ such that PNUDG of G is acyclic. In this section we transform G into a system of process equations G_4 in post-PEGNF (cf. Definition 4.1.7). The resulting system will contain process equations for all process names in $|G|$ with the same types of data parameters involved, as well as, possibly, process equations for other process names. As in the pCRL case, we first apply the preprocessing step of Section 4.2.1, which renames nested bound variables.

5.1.1 Reduction by Simple Rewriting

By applying term rewriting we get an equivalent set of process equations to the given one, but with terms in right-hand sides in the more restricted form as presented in Table 5.1. This step is similar to the one described in Section 4.2.2, but here we have more rewrite rules that are needed to deal with parallel composition and renaming operations, which were not present in the case of pCRL.

The rewrite rules that we apply to the right-hand sides of the equations are listed in Tables 4.2 and 5.2. The symbols $\sum_{d:D}$ are treated in this rewrite system as function symbols, not as binders. This is justified by the fact that we have renamed all nested bound variables, which allows the use of first order term rewriting. The

$$
\begin{aligned}
p &::= p_1 \mid \delta \\
p_1 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid p_1 + p_1 \mid p_2 \cdot p \mid p_1 \parallel p_1 \mid \sum_{d:D} p_3 \mid p_4 \lhd c \rhd \delta \mid \partial_H(p_5) \mid \tau_I(p_6) \\
&\qquad \mid \rho_R(p_7) \\
p_2 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid p_1 + p_1 \mid p_2 \cdot p \mid p_1 \parallel p_1 \mid \partial_H(p_5) \mid \tau_I(p_6) \mid \rho_R(p_7) \\
p_3 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid p_2 \cdot p \mid p_1 \parallel p_1 \mid \sum_{d:D} p_3 \mid p_4 \lhd c \rhd \delta \mid \partial_H(p_5) \mid \tau_I(p_6) \mid \rho_R(p_7) \\
p_4 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid p_2 \cdot p \mid p_1 \parallel p_1 \mid \partial_H(p_5) \mid \tau_I(p_6) \mid \rho_R(p_7) \\
p_5 &::= \mathsf{Y}(\overrightarrow{t}) \mid p_1 \parallel p_1 \\
p_6 &::= p_5 \mid \partial_H(p_5) \\
p_7 &::= p_6 \mid \tau_I(p_6)
\end{aligned}
$$

Table 5.1: Syntax of terms after simple rewriting.

mapping induced by the rewrite rules for a given system of process equations G is called *rewr* : $\mathit{Terms}(|G|) \to \mathit{Terms}(|G|)$.

Before applying rewriting we eliminate all terms of the form $\_ \lhd \_ \rhd \_$ with the third argument different from δ, with the following rule:

$$y \not\equiv \delta \implies x \lhd c \rhd y \to x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta \qquad \text{(RCOND3)}$$

Rewriting is performed modulo the identities presented in Table 5.3.

The optimization rules presented in Table 5.4 are not needed to get the desired restricted syntactic form, but can be used to simplify the terms. They could be applied with higher priority than the rules in Tables 4.2 and 5.2 to achieve possible reductions. Note that the rule (RSCA′) could lead to optimizations only in cases where x is completely guarded, and y or z are not.

Proposition 5.1.1. *The commutative/associative term rewriting system of Tables 4.2 and 5.2 is terminating.*

Proof. We can transform this commutative/associative term rewriting system into a normal one by adding the symmetric rules for the first rule in Table 4.2 and the first two rules in Table 5.2, and directing the associativity axioms. Termination of the obtained system can be proved by constructing the RPO for the following order on the operations:

$$\partial_H > \tau_I > \rho_R > \parallel > \cdot > \_ \lhd c \rhd \delta > \sum > + > \mathsf{a}(\overrightarrow{t}) > \delta$$

Another way of proving termination is by using the AC-RPO technique [90]. □

Lemma 5.1.2. *For any process term p not containing $p_1 \lhd c \rhd p_2$, where $p_2 \not\equiv \delta$, we have that $rewr(p)$ has the syntax defined in Table 5.1.*

Proof. Let $q = rewr(p)$. It can be seen from the rewrite rules that they preserve the syntax (4.1). Suppose q does not satisfy the syntax defined in Table 5.1. All of the possibilities for q that exist imply that q is reducible. Some of the possibilities are shown in the proof of Lemma 4.2.3; for the rest the appropriate rules can be easily found in Table 5.2. □

$x \parallel \delta \to x \cdot \delta$		(RSCD1)
$(x \cdot \delta) \parallel y \to (x \parallel y) \cdot \delta$		(RSCD2)
$\partial_H(\mathsf{a}(\overrightarrow{t})) \to \delta$	if $\mathsf{a} \in H$	(RD2)
$\partial_H(\mathsf{a}(\overrightarrow{t})) \to \mathsf{a}(\overrightarrow{t})$	if $\mathsf{a} \notin H$	(RD1)
$\partial_H(\tau) \to \tau$		(RD1′)
$\partial_H(\delta) \to \delta$		(RD2′)
$\partial_H(x + y) \to \partial_H(x) + \partial_H(y)$		(RD3)
$\partial_H(x \cdot y) \to \partial_H(x) \cdot \partial_H(y)$		(RD4)
$\partial_H\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \partial_H(x)$		(RSUM8)
$\partial_H(x \lhd c \rhd \delta) \to \partial_H(x) \lhd c \rhd \delta$		(RD5)
$\partial_{H_1}(\partial_{H_2}(x)) \to \partial_{H_1 \cup H_2}(x)$		(RDD)
$\partial_H(\tau_I(x)) \to \tau_I(\partial_{H \setminus I}(x))$		(RDT)
$\partial_H(\rho_R(x)) \to \rho_R(\partial_{R^{-1}(H)}(x))$		(RDR)
$\tau_I(\mathsf{a}(\overrightarrow{t})) \to \tau$	if $\mathsf{a} \in I$	(RT2)
$\tau_I(\mathsf{a}(\overrightarrow{t})) \to \mathsf{a}(\overrightarrow{t})$	if $\mathsf{a} \notin I$	(RT1)
$\tau_I(\tau) \to \tau$		(RT2′)
$\tau_I(\delta) \to \delta$		(RT1′)
$\tau_I(x + y) \to \tau_I(x) + \tau_I(y)$		(RT3)
$\tau_I(x \cdot y) \to \tau_I(x) \cdot \tau_I(y)$		(RT4)
$\tau_I\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \tau_I(x)$		(RSUM9)
$\tau_I(x \lhd c \rhd \delta) \to \tau_I(x) \lhd c \rhd \delta$		(RT5)
$\tau_{I_1}(\tau_{I_2}(x)) \to \tau_{I_1 \cup I_2}(x)$		(RTT)
$\tau_I(\rho_R(x)) \to \rho_R(\tau_{R^{-1}(I)}(x))$		(RTR)
$\rho_R(\mathsf{a}(\overrightarrow{t})) \to R(\mathsf{a})(\overrightarrow{t})$		(RR1)
$\rho_R(\tau) \to \tau$		(RRT)
$\rho_R(\delta) \to \delta$		(RRD)
$\rho_R(x + y) \to \rho_R(x) + \rho_R(y)$		(RR3)
$\rho_R(x \cdot y) \to \rho_R(x) \cdot \rho_R(y)$		(RR4)
$\rho_R\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \rho_R(x)$		(RSUM10)
$\rho_R(x \lhd c \rhd \delta) \to \rho_R(x) \lhd c \rhd \delta$		(RR5)
$\rho_{R_1}(\rho_{R_2}(x)) \to \rho_{R_1 \circ R_2}(x)$		(RRR)

Table 5.2: Rewrite rules defining *rewr* (Part 2).

Proposition 5.1.3. *Let G_2 be the result of applying the rewriting to G_1. Then $G_2 = G_1$.*

Proof. Taking into account that G_1 contains no nested occurrences of the same bound variable, each rewrite rule is a consequence of the axioms of μCRL. By Lemma 3.5.1

$$
\begin{aligned}
x + y &\approx y + x \\
x + (y + z) &\approx (x + y) + z \\
(x \cdot y) \cdot z &\approx x \cdot (y \cdot z) \\
x \parallel y &\approx y \parallel x \\
x \parallel (y \parallel z) &\approx (x \parallel y) \parallel z
\end{aligned}
$$

Table 5.3: The rewriting is performed modulo these identities.

$x + x \to x$	(RA3)
$x \lhd c \rhd x \to x$	(RCOND0)
$x \lhd \mathbf{t} \rhd y \to x$	(RCOND1)
$x \lhd \mathbf{f} \rhd y \to y$	(RCOND2)
$x \lhd c_1 \rhd \delta + x \lhd c_2 \rhd \delta \to x \lhd c_1 \vee c_2 \rhd \delta$	(RCOND5)
$(x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \to x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2$	(RSCA)
$x \cdot (y \lhd c \rhd z) \to x \cdot y \lhd c \rhd x \cdot z$	(RSCA$'$)
$\tau_I(\partial_H(x)) \to \tau_{I \setminus H}(\partial_H(x))$	(RTD)
$\rho_R(\tau_I(\partial_H(x))) \to \rho_{R_{I \cup H}}(\tau_I(\partial_H(x)))$	(RRTD)
$\rho_R(\tau_I(x)) \to \rho_{R_I}(\tau_I(x))$	(RRT$'$)
$\rho_R(\partial_H(x)) \to \rho_{R_H}(\partial_H(x))$	(RRD$'$)
$\partial_\emptyset(x) \to x$	(RD0)
$\tau_\emptyset(x) \to x$	(RT0)
$\rho_{R_{ActLab}}(x) \to x$	(RR0)

where $R_S(\mathsf{a})$ for $S \subseteq \mathit{ActLab}$ is defined to be equal to a if $\mathsf{a} \in S$ and to $R(\mathsf{a})$ otherwise.

Table 5.4: Optimization rules.

we get $G_2 = G_1$. □

As a result of applying simple rewriting the number of equations obviously remains the same. The right-hand sides of the equations may grow in a linear fashion with respect to the number of operation symbols of sort *Proc* occurrences. This is because a number of rules copy operation symbols when distributing over $+$ or $\cdot$ (for example the rule (RSUM4) copies the summation symbol). It can be checked that the total number of $+, \cdot$ and $\parallel$ occurrences does not increase during rewriting (except for certain optimization rules). Therefore the number of such copyings is linear in term size. The number of occurrences of action labels and process names does not increase during rewriting.

5.1.2 Adding New Process Equations

This step is similar to the one described in Section 4.2.3, with the only difference that we have more operations, which are treated similarly to sequential composition. We extend the transformations S_1 and S_2 with the following rules:

$$\begin{array}{cc}
S_1(S, p_1 \parallel p_2) \to S_2(S, p_1 \parallel p_2) & S_2(S, p_1 \parallel p_2) \to S_2(S, p_1) \parallel S_2(S, p_2) \\
S_1(S, \partial_H(p)) \to S_2(S, \partial_H(p)) & S_2(S, \partial_H(p)) \to \partial_H(S_2(S, p)) \\
S_1(S, \tau_I(p)) \to S_2(S, \tau_I(p)) & S_2(S, \tau_I(p)) \to \tau_I(S_2(S, p)) \\
S_1(S, \rho_R(p)) \to S_2(S, \rho_R(p)) & S_2(S, \rho_R(p)) \to \rho_R(S_2(S, p))
\end{array}$$

As the result of replacing every equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = p_\mathsf{X}$ of G_2 by $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = S_1(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}, p_\mathsf{X})$ we get a system of process equations G_3, which is in pre-PEGNF.

Proposition 5.1.4. *The functions S_1 and S_2 are well-defined.*

Proof. Using the order on the operations $S_1 > +, S_1 > \sum, S_2 > \cdot, S_2 > \parallel, S_2 > \rho_R, S_2 > \tau_I, S_2 > \partial_H$ it can be shown that infinite recursion is not possible for any admissible arguments given. □

Lemma 5.1.5. *All process equations in G_3 are in pre-PEGNF.*

Proof. Similar to the proof of Lemma 4.2.7. □

Proposition 5.1.6. *For any process name X in G_2 we have $(\mathsf{X}, G_3) = (\mathsf{X}, G_2)$.*

Proof. The statement follows from Lemma 3.5.2. □

Again, as in the pCRL case, the transformation described in this subsection does not increase the size of terms. The number of process equations may increase linearly in the size of terms in the original system.

5.1.3 Guarding

Next we transform the equations of G_3 to PEGNF. To this end, we use the function *guard* : *DVar* × *Terms*(|*G*|) → *Terms*(|*G*|), which replaces unguarded occurrences of process names with the right-hand sides of their defining equations. It is defined as follows:

$$\begin{aligned}
&\mathit{guard}\Big(S, \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} p_i \lhd c_i \rhd \delta\Big) = \mathit{rewr}\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathit{guard}(S \cup \{\overrightarrow{e_i}\}, p_i) \lhd c_i \rhd \delta\Big)\\
&\mathit{guard}(S, \mathsf{a}(\overrightarrow{t})) = \mathsf{a}(\overrightarrow{t})\\
&\mathit{guard}(S, \delta) = \delta\\
&\mathit{guard}(S, \mathsf{Y}(\overrightarrow{t})) = \mathit{guard}\Big(S, S_0(S \setminus \{\mathit{pars}(\mathsf{Y})\}, \mathit{rhs}(\mathsf{Y}))\big[\mathit{pars}(\mathsf{Y}) := \overrightarrow{t}\big]\Big)\\
&\mathit{guard}(S, p_1 \cdot p_2) = \mathit{rewr}\Big(\mathit{simpl}\big(\mathit{guard}(S, p_1) \cdot p_2\big)\Big)\\
&\mathit{guard}(S, \rho_R \circ \tau_I \circ \partial_H(p)) = \mathit{rewr}\big(\rho_R \circ \tau_I \circ \partial_H(\mathit{guard}(S, p))\big)\\
&\mathit{guard}\big(S, p_1 \parallel p_2\big) = \mathit{rewr}\Big(\mathit{simpl}\big(\mathit{guard}(S, p_1) \mathbin{\|\!_} p_2\big) + \mathit{simpl}\big(\mathit{guard}(S, p_2) \mathbin{\|\!_} p_1\big)\\
&\qquad\qquad\qquad\qquad + \mathit{simpl}\big(\mathit{guard}(S, p_1) \mid \mathit{guard}(S, p_2)\big)\Big)
\end{aligned}$$

Here we use the function *rewr* from Subsection 5.1.1 and the function S_0 from Subsection 4.2.1. The function *simpl* is defined as follows:

$$
\begin{aligned}
&simpl\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i})\cdot p_i \lhd c_i \rhd \delta + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \lhd c_j \rhd \delta\Big)\cdot p\Bigg)\\
&\quad = \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i})\cdot (p_i\cdot p) \lhd c_i \rhd \delta + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j})\cdot p \lhd c_j \rhd \delta
\end{aligned}
$$

$$
\begin{aligned}
&simpl\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i})\cdot p_i \lhd c_i \rhd \delta + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \lhd c_j \rhd \delta\Big) \;\underline{\|}\; p\Bigg)\\
&\quad = \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i})\cdot (p_i \parallel p) \lhd c_i \rhd \delta + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j})\cdot p \lhd c_j \rhd \delta
\end{aligned}
$$

$$
\begin{aligned}
&simpl\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta\\
&\qquad\qquad + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta\Big)\\
&\quad \Big|\Big(\sum_{i\in I'}\sum_{\overrightarrow{e_i':E_i'}} \mathsf{a}_i'(\overrightarrow{f_i'}(\overrightarrow{d',e_i'}))\cdot p_i'(\overrightarrow{d',e_i'}) \lhd c_i'(\overrightarrow{d',e_i'}) \rhd \delta\\
&\qquad\qquad + \sum_{j\in J'}\sum_{\overrightarrow{e_j':E_j'}} \mathsf{a}_j'(\overrightarrow{f_j'}(\overrightarrow{d',e_j'})) \lhd c_j'(\overrightarrow{d',e_j'}) \rhd \delta\Big)\Bigg)\\
&\quad = \sum_{(k,l)\in I\gamma I'}\sum_{\overrightarrow{e_k:E_k,e_l':E_l'}} \gamma(\mathsf{a}_k,\mathsf{a}_l')(\overrightarrow{f_k}(\overrightarrow{d,e_k}))\cdot (p_k(\overrightarrow{d,e_k}) \parallel p_l(\overrightarrow{d',e_l'}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f_l'}(\overrightarrow{d',e_l'}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c_l'(\overrightarrow{d',e_l'}) \rhd \delta\\
&\qquad + \sum_{(k,l)\in I\gamma J'}\sum_{\overrightarrow{e_k:E_k,e_l':E_l'}} \gamma(\mathsf{a}_k,\mathsf{a}_l')(\overrightarrow{f_k}(\overrightarrow{d,e_k}))\cdot p_k(\overrightarrow{d,e_k})\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f_l'}(\overrightarrow{d',e_l'}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c_l'(\overrightarrow{d',e_l'}) \rhd \delta\\
&\qquad + \sum_{(k,l)\in J\gamma I'}\sum_{\overrightarrow{e_k:E_k,e_l':E_l'}} \gamma(\mathsf{a}_k,\mathsf{a}_l')(\overrightarrow{f_k}(\overrightarrow{d,e_k}))\cdot p_l(\overrightarrow{d',e_l'})\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f_l'}(\overrightarrow{d',e_l'}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c_l'(\overrightarrow{d',e_l'}) \rhd \delta\\
&\qquad + \sum_{(k,l)\in J\gamma J'}\sum_{\overrightarrow{e_k:E_k,e_l':E_l'}} \gamma(\mathsf{a}_k,\mathsf{a}_l')(\overrightarrow{f_k}(\overrightarrow{d,e_k}))\\
&\qquad\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f_l'}(\overrightarrow{d',e_l'}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c_l'(\overrightarrow{d',e_l'}) \rhd \delta
\end{aligned}
$$

where $P\gamma Q = \{(p,q) \in P \times Q \mid \gamma(\mathsf{a}_p,\mathsf{a}_q') \text{ is defined}\}$. The function *simpl* shows that for any term p^1 and p^2 in the form of a right-hand side of an equation in PEGNF, and for any term p having syntax (4.5) we can transform $p^1\cdot p$, $p^1 \;\underline{\|}\; p$ and $p^1 \mid p^2$ to the form of a right-hand side of an equation in PEGNF by applying the axioms of μCRL.

Proposition 5.1.7. *For any finite system G_3 in pre-PEGNF with acyclic PNUDG, and any process name* X *in it, the function guard is well-defined on* $rhs(\mathsf{X}, G_3)$.

Proof. The proof is similar to the proof of Proposition 4.2.9 in Section 4.2.4. □

We define the system G_4 in the following way. For each equation

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} p_i(\overrightarrow{d, e_i}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta \quad \text{in } G_3 \text{, we add}$$

$$\mathsf{X}(\overrightarrow{d{:}D}) = guard\Big(\{\, \overrightarrow{d}\,\}, \sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} p_i(\overrightarrow{d, e_i}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta\Big) \quad \text{to } G_4.$$

Lemma 5.1.8. *The equations in G_4 are in PEGNF.*

Proof. The proof goes along the lines of the proof of Lemma 4.2.10. Here we use the facts that *simpl* produces PEGNF terms and *rewr* preserves them. □

Proposition 5.1.9. *Let G_3 and G_4 be defined as above. Then $G_3 = G_4$.*

Proof. It was already noted before that the transformations performed by *rewr* and S_0 are derivable from the axioms of μCRL. It is easy to see that the transformations performed by *simpl* are derivable from the axioms as well. Below we list the axioms that are used in the derivations of the three *simpl* cases:

1. (A4), (SUM5) (the variables from $\overrightarrow{e_i}$ and $\overrightarrow{e_j}$ do not occur freely in p), (Cond6), and (A5);

2. (CM4), (SUM6) (the variables from $\overrightarrow{e_i}$ and $\overrightarrow{e_j}$ do not occur freely in p), (Cond8), (CM2) and (CM3);

3. (CM8) and (CM9); (SUM7) and (SUM7′) (here we use the fact that the following three sets: bound variables of the first argument of *simpl*, bound variables of the second argument of *simpl*, free variables of both arguments, are pairwise disjoint); (Cond9) and (Cond9′); (Cond4); (CM5), (CM6) and (CM7); (CF1), (Cond6) and (Cond4) (or (CF2), (A7), Lemma 2.2.6.1, (SUM1) and (A6) in case the actions do not communicate).

In case one of the PEGNF arguments of *simpl* is δ, the axioms used are (A7), (CM2), (SCD2) and (SC3). According to Lemma 3.5.3 and Lemma 3.5.1 all transformations performed by *guard* lead to equivalent systems. We note that care has been taken to rename some data variables during the substitution (in the third clause of the *guard* definition) in order to make the substitution and the following applications of the axioms sound. □

The transformation performed in this step does not increase the number of equations, but their sizes may grow exponentially, due to application of (A4). (See an example of such an exponential growth in Section 4.2.4.) We also note that similar growth is possible due to application of axioms (CM4) for the left merge, and (CM8) and (CM9) for communication. In cases with multi-party communication we do not need to have n equations, as in the pCRL example, to achieve this growth. Having one equation with n recursive calls is sufficient.

Example 5.1.10. Let n be a natural number and let the system of process equations G contain the following two equations.

$$\begin{aligned}
&\mathsf{Y} = \mathsf{X}(0) \parallel \cdots \parallel \mathsf{X}(n)\\
&\mathsf{X}(n{:}\mathit{Nat}) = \mathsf{a} + \mathsf{b}(n)
\end{aligned}$$

We further assume that $\gamma(\mathsf{a},\mathsf{a}) = \mathsf{a}$, and γ is undefined for the other arguments. By induction on n it is not difficult to show that after applying guarding we get the system of equations which is equivalent to the following one:

$$\begin{aligned}
\mathsf{Y} = \mathsf{a} + &\sum_{\{i_0,\dots,i_k\}\in 2^{\{0,\dots,n\}}\setminus\{\emptyset,\{0,\dots,n\}\}} \mathsf{a}\cdot(\mathsf{X}(i_0)\parallel\cdots\parallel\mathsf{X}(i_n))\\
&+ \sum_{m\in\{0,\dots,n\}} \mathsf{b}(m)\cdot(\mathsf{X}(0)\parallel\cdots\parallel\mathsf{X}(m-1)\parallel\mathsf{X}(m+1)\parallel\cdots\parallel\mathsf{X}(n))\\
\mathsf{X}(n{:}\mathit{Nat}) &= \mathsf{a} + \mathsf{b}(n)
\end{aligned}$$

This example shows that the term in the right-hand side of the equation for Y contains more than 2^{n+1} summands after the transformation.

5.1.4 Postprocessing

In this subsection we transform all equations of G_4 into post-PEGNF. It is done in a similar way as described in Section 4.2.5, and the possible optimizations mentioned there (regular linearization process) also apply to the settings of full μCRL. In order to do this transformation, we need to eliminate all actions and δ that appear in terms p_i in PEGNF. This is achieved by introducing a new process name $\mathsf{X_a}$ for each action a that occurs inside the process terms p_i, with parameters corresponding to those of the action (and a new process name X_δ for δ). Thus we add equations $\mathsf{X_a}(\overrightarrow{d_\mathsf{a}{:}D_\mathsf{a}}) = \mathsf{a}(\overrightarrow{d_\mathsf{a}})$ and $\mathsf{X}_\delta = \delta$ to the system, and replace the occurrences of actions $\mathsf{a}(\overrightarrow{t})$ by $\mathsf{X_a}(\overrightarrow{t})$, and δ by X_δ.

Proposition 5.1.11. *Let the system G_5 of process equations be obtained after post-processing the system G_4 as described above. Then for all $\mathsf{X}\in|G_4|$ we have $(\mathsf{X},G_5) = (\mathsf{X},G_4)$ and G_5 is in post-PEGNF.*

Proof. According to Lemma 3.5.2 this transformation is correct and leads to a system that obviously is in PEGNF. $\square$

It is also possible to eliminate renaming, hiding and encapsulation operations that do not have parallel composition in their arguments by introducing more terms of the form $\rho_R(\tau_I(\partial_H(p_1\parallel p_2)))$, thus removing $\rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}))))$ from the grammar (4.8). This can be done by introducing a fresh process name Z for every different $\rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}))))$ together with the defining equation $\mathsf{Z}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}) = \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}))))$. By taking the $\mathit{rhs}(\mathsf{Y})$ and applying the rewrite rules for renaming operators we either get rid of the construct, or get a new instance of it, possibly with different R, I, and/or H. Given the fact that the set of actions is finite, the number of different R,

I, and H is also finite, and therefore we cannot introduce an infinite number of fresh process names in this way.

An open question is whether we can eliminate $\rho_R(\tau_I(\partial_H(p_1 \parallel p_2)))$ by introducing more process equations and renamings of the form $\rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}))))$. An interesting example would be $\mathsf{X} = \mathsf{a} \cdot \partial_{\{\mathsf{b}\}}(\mathsf{X} \parallel \partial_{\{\mathsf{b}\}}(\mathsf{X} \parallel \mathsf{X}))$ with $\gamma(\mathsf{a}, \mathsf{a}) = \mathsf{a}$.

It remains an interesting question whether all renaming operations can be eliminated without the use of infinite data types. We conjecture that it is not possible. The partial elimination of renaming operators do not lead to simplifications of the data type that we need to encode. Total elimination of renaming operations would provide such a simplification.

Summarizing, the initial and the current μCRL specification are related by $(\mathsf{X}, G) = (\mathsf{X}, G_5)$, and we have not added any extra data type definitions to the current specification up to now.

5.2 Introduction of Lists-of-Multisets

Like in the pCRL case, we now collapse all of the equations in G_5 into a single one (cf. Section 4.3), obtaining a system G_7, which contains this equation only. The final step in the linearization of μCRL processes consists of the introduction of a data parameter, that allows to model sequential and parallel compositions of process names with parameters, as a single process term. The data parameter should also encode renaming, hiding and encapsulation operations. In the case that no such sequential or parallel composition occurs in the equation, we do not apply this step. The renaming, hiding and encapsulation operations can, in this case, be eliminated using the transformation described in Section 5.1.4. We note that if no parallel composition operations were present, we could also eliminate the renaming, hiding and encapsulation operations and arrive at the pCRL case. In the case of pCRL processes the data type needed was a stack (cf. Section 4.4). The case of μCRL is more complicated in the following ways.

- Parallel composition is present in addition to sequential composition.
- Instead of a single process that was ready to be executed in the sequential case, we can have many parallel components represented by their state vectors, and the number of components can change during process execution.
- The components may communicate; thus simultaneous execution of two (handshaking) or more (multi-party communication) components is possible.
- The renaming, hiding and encapsulation operations can influence the way in which a component (or more than one of them) can be executed.

As a first step we consider the case with handshaking and no renaming, hiding and encapsulation operations; after that we add these operations, and finally outline the multi-party communication case. This is done in order to divide the explanation of the data type into smaller and more understandable parts. In addition to that, for each particular specification the appropriate data type can be used, depending on the presence of renaming operations and the type of communication used.

5.2.1 Parallel and Sequential Compositions with Handshaking

Assuming that no renaming operators are present, let G_7 contain a single μCRL process equation in post-PEGNF:

$$\begin{aligned}\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta \\ &+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta\end{aligned} \tag{5.1}$$

where $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$p ::= \mathsf{X}(\overrightarrow{t}) \mid p\cdot p \mid p \,\|\, p \tag{5.2}$$

The form above differs from an LPE in having sequential and parallel compositions of recursive calls instead of a single recursive call. We define the data type *State* (Appendix C.1) to represent the state vector $\overrightarrow{d{:}D}$. It is a simple tuple data type, that has a constructor $\mathit{state} : \overrightarrow{D} \to \mathit{State}$, projection functions $\mathit{pr}_i : \mathit{State} \to D_i$, equality predicate, if-then-else construction, and a greater-than predicate *gt*.[1]

The data type *LM* is used to represent a list containing state vectors $\overrightarrow{d}$ and/or multisets of elements of type *LM*. For the latter multisets we use the data type *ML* (see Appendix C.2 for the implementation details). The main idea is to represent a number of consecutive sequential compositions as a list, and a number of consecutive parallel compositions as a multiset. These lists and multisets can be nested up to arbitrary depth, as the terms can contain arbitrarily nested parallel and sequential compositions. A single state vector is represented as the list containing it. Thus the sort *LM* has three constructors:

- $\mathit{LM0} :\to \mathit{LM}$, representing the empty list,
- $\mathit{seq1} : \mathit{State} \times \mathit{LM} \to \mathit{LM}$, with $\mathit{seq1}(d, \mathit{lm})$ representing the list with the state vector d added as the head of lm,
- $\mathit{seqM} : \mathit{ML} \times \mathit{LM} \to \mathit{LM}$, with $\mathit{seqM}(\mathit{ml}, \mathit{lm})$ representing the list with the multiset ml added as the head of lm,

and the sort *ML* has two constructors:

- $\mathit{ML} : \mathit{LM} \to \mathit{ML}$, representing the multiset containing one list lm,
- $\mathit{par} : \mathit{LM} \times \mathit{ML} \to \mathit{ML}$, with $\mathit{par}(\mathit{lm}, \mathit{ml})$ representing the multiset with the list lm added to ml.

We note however, that with these constructors we can have different terms representing the same semantical value. For instance the following equivalent terms can be identified using the definitions in Appendix C.2:

[1] In the text, often we do not distinguish between $\overrightarrow{D}$ and *State*, and do not use *state* and pr_i, but use vector notation instead.

- $seqM(ML(LM0), lm) \approx lm$,
- $seqM(ML(seq1(d, lm1)), lm) \approx seq1(d, conc(lm1, lm))$,
- $seqM(ML(seqM(ml, lm1)), lm) \approx seqM(ml, conc(lm1, lm))$,
- $ML(seqM(ml, LM0)) \approx ml$,
- $par(LM0, ml) \approx ml$,
- $par(lm, ML(lm1)) \approx par(lm1, ML(lm))$,
- $par(seqM(ml, LM0), ml1) \approx comp(ml, ml1)$,

where the functions *conc* and *comp* are explained below. The first three identities are due to the fact that a multiset at the left-hand side of a sequential composition is only needed if it contains at least two elements. The fourth identity says that putting a multiset into a list and then putting this list into a multiset does not change anything. The sixth identity is due to commutativity of parallel composition. The fifth and the last one say that a list at the left-hand side of a parallel composition is only needed if it contains at least two elements.

There are more such identities, and we want to operate with the right-hand sides of these identities only. We define the *normal forms* for lists and multisets in the following way. A term of sort *LM* is in normal form if it is in one of the following three forms:

- $LM0$,
- $seq1(d, lm)$,
- $seqM(ml, lm)$,

where

- d is a term of sort *State*,
- lm is a term of sort *LM* in normal form,
- ml is a term of sort *ML* in normal form having *par* as its outermost symbol.

A term of sort *ML* is in normal form if it is in one of the following two forms:

- $ML(lm)$,
- $par(lm_1, \ldots par(lm_n, ML(lm_{n+1})) \ldots)$,

where for all $i \in \{1, \ldots, n+1\}$:

- lm, lm_i are terms of sort *LM* in normal form, and not of the form $seqM(ml, LM0)$,
- $lm_i \neq LM0$,
- $\neg gt(lm_i, lm_{i+1})$.

The gt function (greater than) is defined on LM and ML using the function gt on the sort $State$.

Preservation of normal forms is achieved by defining auxiliary functions that guarantee the generation of normal forms only, if the arguments are in normal forms:

- $conc : LM \times LM \to LM$,
- $conp : ML \times LM \to LM$,
- $mkml : LM \to ML$,
- $comp : ML \times ML \to ML$.

The first one is used to concatenate two lists. The second – to prepend a multiset to a list. The third – to make a multiset out of a list, and the last one – to concatenate two multisets. The implementation of these functions can be found in Appendix C.2. It can be shown by induction that if the arguments of the auxiliary functions are in normal form, then the result also rewrites to a term in normal form. In addition, this property can be shown for all functions in Appendix C that generate terms of sort LM or ML.

Preservation of normal forms gives us a simple way to define equality on the LM and ML data types. We can also check that the following properties are preserved for any lm and ml in normal form:

- $mkml(conp(ml, LM0)) \approx ml$,
- $conp(mkml(lm), LM0) \approx lm$.

We use the functions $seqc$ and $parc$ to represent sequential and parallel compositions on the sort LM, respectively. The following properties of these functions can be checked, under the assumption that all arguments are in normal form: associativity of $seqc$, associativity and commutativity of $parc$, $LM0$ is zero element for both functions.

For each term p_i from equation (5.1) we construct the term $\mathbf{mklm}_i[p_i] : State \times \overrightarrow{E_i} \to LM$, which gives us a way to represent the terms p_i as the terms of sort LM, in the following way:

$$\begin{aligned}
&\mathbf{mklm}_i[\mathsf{X}(\overrightarrow{t})](\overrightarrow{t_d, t_{e_i}}) = seq1(\overrightarrow{t}[\overrightarrow{d, e_i} := \overrightarrow{t_d, t_{e_i}}], LM0)\\
&\mathbf{mklm}_i[p^1 \cdot p^2](\overrightarrow{t_d, t_{e_i}}) = seqc(\mathbf{mklm}_i[p^1](\overrightarrow{t_d, t_{e_i}}), \mathbf{mklm}_i[p^2](\overrightarrow{t_d, t_{e_i}}))\\
&\mathbf{mklm}_i[p^1 \parallel p^2](\overrightarrow{t_d, t_{e_i}}) = parc(\mathbf{mklm}_i[p^1](\overrightarrow{t_d, t_{e_i}}), \mathbf{mklm}_i[p^2](\overrightarrow{t_d, t_{e_i}}))
\end{aligned}$$

As an example, if $p_i = (\mathsf{X}(n) \parallel \mathsf{X}(s(n))) \cdot \mathsf{X}(s(s(n)))$, then

$$\mathbf{mklm}_i[p_i](n) = \langle\{n, s(n)\}, s(s(n))\rangle$$

(or as a term $seqM(par(seq1(n, LM0), ML(seq1(s(n), LM0)), seq1(s(s(n)), LM0))))$.

As explained earlier, the data type LM represents a nesting of sequential and parallel compositions of the state vectors of process X defined by equation (5.1). For a given $lm{:}LM$, an important notion is the multiset of the state vectors of X that are *ready to be executed*. In other words, these are state vectors of X that are not prepended by other state vectors of X with a sequential composition. We call this multiset of state vectors of X from lm the *first layer* of lm. More formally, an occurrence of $d{:}State$ belongs to the *first layer* of lm if lm has no subterm of the form $seq1(d_1, lm_1)$ or $seqM(ml_1, lm_1)$ such that this occurrence of d is in lm_1.

The following functions involving the notion of the first layer are used in the definitions of the resulting LPE:

$lenf : LM \to Nat$	– the number of elements in the first layer
$getf1 : LM \times Nat \to \overrightarrow{D}$	– get n-th element
$replf1 : LM \times Nat \times LM \to LM$	– replace n-th element with an lm
$remf1 : LM \times Nat \to LM$	– remove n-th element
$replf2 : LM \times Nat \times Nat \times LM \times LM \to LM$	– replace two elements
$replremf2 : LM \times Nat \times Nat \times LM \to LM$	– replace one and remove the other element
$remf2 : LM \times Nat \times Nat \to LM$	– remove two elements

As can be seen from the implementation (Appendix C.2), removing an element from an $lm{:}LM$ is equivalent to replacing it with $LM0$. In the example considered earlier, we have two elements in the first layer, where n has number zero, and $s(n)$ has number one.

Assume the system G_7 consists of process equation X as defined in (5.1). We can now define a system L consisting of process equation Z, that mimics the behavior of X, in the following way:

$$
\begin{aligned}
&\mathsf{Z}(lm{:}LM) = \\
&\quad \sum_{i\in I}\sum_{n:Nat}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{getf1(lm,n),e_i}))\cdot \mathsf{Z}(replf1(lm,n,\mathbf{mklm}_i[p_i](\overrightarrow{getf1(lm,n),e_i}))) \\
&\qquad\qquad \lhd n < lenf(lm) \land c_i(\overrightarrow{getf1(lm,n),e_i}) \rhd \delta \\
&+\sum_{j\in J}\sum_{n:Nat}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{getf1(lm,n),e_j}))\cdot \mathsf{Z}(remf1(lm,n)) \\
&\qquad\qquad \lhd n < lenf(lm) \land remf1(lm,n) \neq \langle\rangle \land c_j(\overrightarrow{getf1(lm,n),e_j}) \rhd \delta \\
&+\sum_{j\in J}\sum_{n:Nat}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{getf1(lm,n),e_j})) \\
&\qquad\qquad \lhd n < lenf(lm) \land remf1(lm,n) = \langle\rangle \land c_j(\overrightarrow{getf1(lm,n),e_j}) \rhd \delta \\
&+ \sum_{(k,l)\in I\gamma I}\sum_{n:Nat}\sum_{m:Nat}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e_l':E_l}} \gamma(\mathsf{a}_k,\mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k})) \\
&\quad \cdot \mathsf{Z}(replf2(lm,n,m,\mathbf{mklm}_k[p_k](\overrightarrow{getf1(lm,n),e_k}),\mathbf{mklm}_l[p_l](\overrightarrow{getf1(lm,m),e_l'}))) \\
&\qquad \lhd n < m \land m < lenf(lm) \land \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e_l'}) \\
&\qquad \land c_k(\overrightarrow{getf1(lm,n),e_k}) \land c_l(\overrightarrow{getf1(lm,m),e_l'}) \rhd \delta
\end{aligned}
$$

$$+ \sum_{(k,l)\in I\gamma J} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \gamma(\mathsf{a}_k, \mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}))$$
$$\cdot\, \mathsf{Z}(replremf2(lm,n,m,\mathbf{mklm}_k[p_k](\overrightarrow{getf1(lm,n),e_k})))$$
$$\lhd\, n \neq m \wedge n < lenf(lm) \wedge m < lenf(lm)$$
$$\wedge\, \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})$$
$$\wedge\, c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l}) \rhd \delta$$
$$+ \sum_{(k,l)\in J\gamma J} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \gamma(\mathsf{a}_k, \mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k})) \cdot \mathsf{Z}(remf2(lm,n,m))$$
$$\lhd\, n < m \wedge m < lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})$$
$$\wedge\, c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l}) \wedge remf2(lm,n,m) \neq \langle\rangle \rhd \delta$$
$$+ \sum_{(k,l)\in J\gamma J} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \gamma(\mathsf{a}_k, \mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}))$$
$$\lhd\, n < m \wedge m < lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})$$
$$\wedge\, c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l}) \wedge remf2(lm,n,m) = \langle\rangle \rhd \delta$$

where $P\gamma Q = \{(k,l) \in P \times Q \mid \gamma(\mathsf{a}_k, \mathsf{a}_l) \text{ is defined}\}$.

The first three sets of summands of the equation represent the singular executions of the ready components (elements of the first layer), which are sometimes called interleavings. The process $\mathsf{Z}(lm)$ can execute any action the original process $\mathsf{X}(\overrightarrow{d})$ can execute, provided that $\overrightarrow{d}$ belongs to the first layer of lm. After that the state of Z becomes lm with the first layer occurrence of $\overrightarrow{d}$ replaced by the LM representation of the resulting parallel/sequential composition generated from the terms p_i taken from the equation for X. The second and third sets represent the case where the ready component terminates. In this case we remove the component from lm and, depending on whether this was the last element of lm, either terminate, or not.

The last four sets of summands represent the dual executions of the ready components by means of synchronous communication of them, sometimes called handshakings. Here we take two different ready components, say $\overrightarrow{d}$ and $\overrightarrow{d'}$ and execute the actions that $\mathsf{X}(\overrightarrow{d})\,|\,\mathsf{X}(\overrightarrow{d'})$ could execute. These are the actions that communicate and have equal parameter vectors. Due to commutativity of the communication function and parallel composition, it is enough to consider only ordered pairs of elements of the first layer (that is why the condition $n < m$ is present if both components perform terminating actions of X, or both do not). In order to determine the next state of Z, we either replace both components by the future behavior of both $\mathsf{X}(\overrightarrow{d})$ and $\mathsf{X}(\overrightarrow{d'})$, respectively (fourth set of summands), or replace one and remove the other (fifth set), or remove both components (last two sets). The last two sets of summands only differ in the fact that the first one does not terminate, and the second one does. This behavior is determined on the fact whether or not the two communicating components were the last two elements of lm.

The following theorem states the correctness of our construction.

Theorem 5.2.1. $(\mathsf{X}(\overrightarrow{d}), G_7) \overset{ind}{\Rightarrow} (\mathsf{Z}(\mathit{seq1}(\overrightarrow{d}, \mathit{LM0})), L)$.

Proof (Sketch). The statement can be proved similarly to Proposition 4.4.1 in Section 4.4. Here we define $w_{\mathsf{Z}}(lm)$ for all well-defined closed terms of sort LM in the following way:

$$
\begin{aligned}
&w_{\mathsf{Z}}(\mathit{LM0}) = \delta\\
&w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t}, \mathit{LM0})) = \mathsf{X}(\overrightarrow{t})\\
&w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t}, \mathit{seq1}(\overrightarrow{t'}, lm))) = \mathsf{X}(\overrightarrow{t}) \cdot w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t'}, lm))\\
&w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t}, \mathit{seqM}(ml, lm))) = \mathsf{X}(\overrightarrow{t}) \cdot w_{\mathsf{Z}}(\mathit{seqM}(ml, lm))\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{ML}(lm_2)), \mathit{LM0})) = w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(lm_2)\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{ML}(lm_2)), \mathit{seq1}(\overrightarrow{t}, lm)))\\
&\qquad\qquad = (w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(lm_2)) \cdot w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t}, lm))\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{ML}(lm_2)), \mathit{seqM}(ml, lm)))\\
&\qquad\qquad = (w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(lm_2)) \cdot w_{\mathsf{Z}}(\mathit{seqM}(ml, lm))\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{par}(lm_2, ml_1)), \mathit{LM0}))\\
&\qquad\qquad = w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_2, ml), \mathit{LM0}))\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{par}(lm_2, ml_1)), \mathit{seq1}(\overrightarrow{t}, lm)))\\
&\qquad\qquad = (w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_2, ml), \mathit{LM0}))) \cdot w_{\mathsf{Z}}(\mathit{seq1}(\overrightarrow{t}, lm))\\
&w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_1, \mathit{par}(lm_2, ml_1)), \mathit{seqM}(ml, lm)))\\
&\qquad\qquad = (w_{\mathsf{Z}}(lm_1) \parallel w_{\mathsf{Z}}(\mathit{seqM}(\mathit{par}(lm_2, ml), \mathit{LM0}))) \cdot w_{\mathsf{Z}}(\mathit{seqM}(ml, lm))
\end{aligned}
$$

The proof for the first case is trivial from the facts that $\mathit{lenf}(\mathit{LM0}) \approx 0$ and $n < 0 \approx \mathbf{f}$. The second, third and fourth cases are very close to the cases in Proposition 4.4.1. They follow from the above identity, the axioms of μCRL and the properties of the data types defined in Appendix C, and the fact that for all the terms p_i from the equation (5.1)

$$w_{\mathsf{Z}}(\mathbf{mklm}_i[p_i](\overrightarrow{t})) \approx p_i[\overrightarrow{d, e_i} := \overrightarrow{t}]$$

In the remaining six cases we have two groups, three cases each. In both groups the second and the third cases follow from the first ones. To prove the fifth case we use the induction hypothesis and assume that $w_{\mathsf{Z}}(lm_1)$ is equal to the right-hand side of $\mathsf{Z}(lm_1)$ with Z replaced by the terms w_{Z}, and the similar fact for $w_{\mathsf{Z}}(lm_2)$. We also use the fact that the communication is limited to handshaking. The eights case can be proved in a similar way. □

5.2.2 Renaming Operators

In this subsection we still assume that only handshaking communication is possible, but allow renaming operations to be present. Taking into account that $x \approx$

$\rho_{R_{ActLab}}(\tau_\emptyset(\partial_\emptyset(x)))$, where R_{ActLab} is the identity mapping, we assume that G_7 contains a single μCRL process equation in post-PEGNF of the following form:

$$\begin{aligned}\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta\\ &+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta\end{aligned} \tag{5.3}$$

where $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$p ::= p\cdot p \mid \rho_R(\tau_I(\partial_H(\mathsf{X}(\overrightarrow{t})))) \mid \rho_R(\tau_I(\partial_H(p \parallel p))) \tag{5.4}$$

We reuse the *State* data type defined in the previous subsection and extend the *LM* and *ML* data types to contain information about renaming operations surrounding a recursive call or a parallel composition, which we call *annotation* (cf. Appendix C.3).

To capture the annotations in the form of a data type, we first need to turn actions into a data type. Let the set of action labels *ActLab* be equal to $\{\mathsf{a}_0,\ldots,\mathsf{a}_n\}$. We define the data types *Act*, *ActSet*, *ActMap* and *Annote* (cf. Appendix C.3), to represent actions, sets of actions, mappings of actions, and triples (R, I, H), respectively. For each action label $\mathsf{a} \in \mathit{ActLab}$ we define $\mathbf{mka}[\mathsf{a}] :\rightarrow \mathit{Act}$ to be equal to $a(i)$, where i is such that $\mathsf{a} = \mathsf{a}_i$. For each $S \subseteq \mathit{ActLab}$ we define $\mathbf{mkas}[S] :\rightarrow \mathit{ActSet}$ such that $\mathbf{mkas}[\{\mathsf{a}_0,\ldots,\mathsf{a}_m\}] = \mathit{add}(\mathbf{mka}[\mathsf{a}_0],\ldots \mathit{add}(\mathbf{mka}[\mathsf{a}_m], \mathit{ActSet0})\ldots)$. For every well-defined action renaming function R (cf. Definition 2.1.4) we define $\mathbf{mkam}[R] :\rightarrow$ *ActMap* to have the property that for any action $\mathsf{a} \in \mathit{Act}$ $\mathit{appl}(\mathbf{mka}[\mathsf{a}], \mathbf{mkam}[R]) = \mathbf{mka}[R(\mathsf{a})]$, where $\mathit{appl} : \mathit{Act} \times \mathit{ActMap} \rightarrow \mathit{Act}$ gives the result of application of a mapping to an action label.

The data types *ALM* (annotated *LM*) and *AML* (annotated *ML*) have the same constructors as *LM* and *ML*, respectively, with the following two type differences that concern the annotations:

- $\mathit{seq1} : \mathit{Annote} \times \mathit{State} \times \mathit{ALM} \rightarrow \mathit{ALM}$, with $\mathit{seq1}(\mathit{ann}, d, \mathit{lm})$ representing the list with the state vector d, annotated with *ann*, added to the head of *lm*,
- $\mathit{par} : \mathit{Annote} \times \mathit{ALM} \times \mathit{AML} \rightarrow \mathit{AML}$, with $\mathit{par}(\mathit{ann}, \mathit{lm}, \mathit{ml})$ representing the multiset with the list *lm* added to *ml* and this parallel composition annotated with *ann*.

Normal forms of the *ALM* and *AML* terms are defined as follows. A term of sort *ALM* is in normal form if it is of the form:

- *ALM0*,
- $\mathit{seq1}(\mathit{ann}, d, \mathit{lm})$,
- $\mathit{seqM}(\mathit{ml}, \mathit{lm})$,

where

- d is a term of sort *State* and *ann* is a term of sort *Annote*,
- lm is a term of sort *ALM* in normal form,
- ml is a term of sort *AML* in normal form having *par* as outermost symbol.

A term of sort *AML* is in normal form if it is of the form:

- $AML(lm)$,
- $par(ann_1, lm_1, \ldots par(ann_n, lm_n, AML(lm_{n+1})) \ldots)$,

where for all $i \in \{1, \ldots, n+1\}$:

- lm, lm_i are terms of sort *ALM* in normal form, not of the form $seqM(par(Ann0, lm', ml), ALM0)$,
- $lm_i \neq ALM0$,
- lm_n is not of the form $seqM(ml, ALM0)$,
- $\neg gt(lm_i, lm_{i+1})$.

The *gt* function (greater than) is defined on *ALM* and *AML* using the functions *gt* on the sorts *State* and *Annote*.

As in the case without annotations, normal forms are preserved by the auxiliary functions *conc*, *conp*, *mkml* and *comp*. In addition to that we have the function *annote* to emulate the application of the renaming operations to an *ALM*. The preservation of normal forms can be shown for all functions that generate terms of sort *ALM* or *AML*. Also, the properties of combinations of *mkml* and *conp*, as well as the properties of *seqc* and *parc* compositions are also valid in the setting with annotations. It is also easy to check that *annote* distributes over *seqc*.

For each term p_i from the equation for X we construct the term $\mathbf{mklm}_i[p_i] : State \times \overrightarrow{E_i} \to ALM$ in the following way:

$$\begin{aligned}
&\mathbf{mklm}_i[\rho_R(\tau_I(\partial_H(\mathsf{X}(\overrightarrow{t}))))](\overrightarrow{t_d, t_{e_i}}) = \\
&\qquad seq1(ann(\mathbf{mkam}[R], \mathbf{mkas}[I], \mathbf{mkas}[H]), \overrightarrow{t}[\overrightarrow{d, e_i} := \overrightarrow{t_d, t_{e_i}}], LM0) \\
&\mathbf{mklm}_i[p^1 \cdot p^2](\overrightarrow{t_d, e_i}) = seqc(\mathbf{mklm}_i[p^1](\overrightarrow{t_d, t_{e_i}}), \mathbf{mklm}_i[p^2](\overrightarrow{t_d, t_{e_i}})) \\
&\mathbf{mklm}_i[\rho_R(\tau_I(\partial_H(p^1 \parallel p^2)))](\overrightarrow{t_d, t_{e_i}}) = \\
&\qquad parc(ann(\mathbf{mkam}[R], \mathbf{mkas}[I], \mathbf{mkas}[H]), \mathbf{mklm}_i[p^1](\overrightarrow{t_d, t_{e_i}}), \mathbf{mklm}_i[p^2](\overrightarrow{t_d, t_{e_i}}))
\end{aligned}$$

As an example, if $p_i = \rho_R(\partial_H(\mathsf{X}(n) \parallel \mathsf{X}(s(n)))) \cdot \tau_I(\partial_{H_1}(\mathsf{X}(s(s(n)))))$, then

$$\begin{aligned}
&\mathbf{mklm}_i[p_i](n) = seqM(par(ann(\mathbf{mkam}[R], ActSet0, \mathbf{mkas}[H]), \\
&seq1(Ann0, n, LM0), ML(seq1(Ann0, s(n), LM0))), \\
&\qquad\qquad seq1(ann(ActMap0, \mathbf{mkas}[I], \mathbf{mkas}[H_1]), s(s(n)), LM0))
\end{aligned}$$

For the precise definition of the *ALM* and *AML* data types we refer to Appendix C.3.

The notion of the first layer is preserved for the case with annotations, but in addition to the state vector, each element of the first layer has its individual annotation, which is a composition of all annotations in the scope of which it appears. In case we are interested in a pair of state vectors from the first layer, we have to consider three annotations. For example (considering just the encapsulations), $\partial_H(\partial_{H_1}(\mathsf{X}(1)) \parallel \partial_{H_2}(\mathsf{X}(2)))$ leads to the pair of the first layer elements (1 and 2), and three annotations (H, H_1, and H_2). The following additional functions involving the notions of the first layer and annotations are used in the definition of the resulting LPE:

$getf1d : ALM \times Nat \to \overrightarrow{D}$	– get n-th element
$getf1a : ALM \times Nat \to Annote$	– get n-th element's annotation
$getf2a0 : ALM \times Nat \times Nat \to Annote$	– get n-th element's annotation up to the junction with the m-th element
$getf2a1 : ALM \times Nat \times Nat \to Annote$	– get m-th element's annotation up to the junction with the n-th element
$getf2a : ALM \times Nat \times Nat \to Annote$	– get (n,m)-th elements' annotation (from the junction upwards)

And as in the case without annotations, removing an element is equivalent to replacing it with *ALM0*.

Assume the system G_7 consists of process equation X as defined in (5.3). A system L consisting of process equation Z, which mimics behavior of X, is defined in Appendix A.1. The following theorem states the correctness of our construction.

Theorem 5.2.2. $(\mathsf{X}(\overrightarrow{d}), G_7) \overset{ind}{\Rightarrow} (\mathsf{Z}(seq1(Anno0, \overrightarrow{d}, LM0)), L)$.

5.2.3 Multi-Party Communication

In this subsection we define the LPE for the case when an arbitrary number of parallel components can be executed synchronously. The number is unknown a priori and is only bound by the number of the elements of the first layer in a particular state. On the other hand, the number of different action labels is finite, and, as will be shown later, so is the number of possible communication configurations.

We start from the simpler sub-case where no renaming operations are present. First of all we introduce some abbreviations to simplify dealing with the commutative associative partial communication function γ. We assume that $\mathsf{e} \notin ActLab$ is such that for any $\mathsf{a} \in ActLab$ $\gamma(\mathsf{a}, \mathsf{e}) = \mathsf{a}$, and recall that $\gamma(\tau, \mathsf{a})$ is undefined. Moreover, taking associativity of γ into account, we define $\gamma(\mathsf{a}_1, \ldots, \mathsf{a}_n) = \gamma(\mathsf{a}_1, \ldots \gamma(\mathsf{a}_{n-1}, \mathsf{a}_n) \ldots)$. For any action label $\mathsf{a} \in ActLab$, we define $\mathsf{a}^0 = \mathsf{e}$, $\mathsf{a}^1 = \mathsf{a}$, and $\mathsf{a}^{n+1} = \gamma(\mathsf{a}, \mathsf{a}^n)$. Similarly, $\tau^0 = \mathsf{e}$, $\tau^1 = \tau$, and τ^n is undefined for all $n > 1$. From the finiteness of *ActLab* it can easily be seen that for any action $\mathsf{a} \in ActLab$ there are minimal natural numbers $p(\mathsf{a})$ (prefix of a) and $c(\mathsf{a})$ (cycle of a) such that the sequence a^n repeats itself after $p(\mathsf{a})$ steps with period $c(\mathsf{a})$. More precisely, taking into account that a^n may become undefined for some n and all greater powers, we define the numbers $p(\mathsf{a})$ and $c(\mathsf{a})$ as follows:

$$p(\mathsf{a}) = \mathbf{min}\{n \in \mathbb{N} \mid \mathsf{a}^n \text{ is undefined } \vee \ \exists m{>}n \ \mathsf{a}^n = \mathsf{a}^m\}$$

$$c(\mathsf{a}) = \begin{cases} 0 & \text{if } \mathsf{a}^{p(\mathsf{a})} \text{ is undefined} \\ \mathbf{min}\{n \in \mathbb{N} \mid n{>}0 \ \wedge \ \mathsf{a}^{p(\mathsf{a})} = \mathsf{a}^{p(\mathsf{a})+n}\} & \text{otherwise} \end{cases}$$

which means that if a^n is undefined for some n, then $p(\mathsf{a})$ is minimal with respect to such n, and in this case we put $c(\mathsf{a}) = 0$. In accordance to this, we define $p(\tau) = 2$ and $c(\tau) = 0$.

Considering the equation for X as defined in (5.1), we take the sets of indices I and J and for all $i \in I \cup J$ we define $p(i) = p(\mathsf{a}_i)$ and $c(i) = c(\mathsf{a}_i)$. For this equation for X we define a notion of *configuration* as a function $\mathit{conf} : I \cup J \to \mathbb{N}$. A particular configuration specifies how many occurrences of an action label take part in a communication. We consider only the configurations that for each action label a_i have no more than $p(i) + c(i) - 1$ occurrences. Moreover we only consider the configurations that are defined. Assuming that $\gamma(\mathit{conf}) = \gamma(\mathsf{a}_0^{\mathit{conf}(0)}, \ldots, \mathsf{a}_m^{\mathit{conf}(m)})$, where $I \cup J = \{0, \ldots, m\}$, we define the set of configurations in the following way:

$$\begin{aligned} \mathit{Conf} = \{\mathit{conf} \mid \forall i \in I \cup J \ \big(0 \leq \mathit{conf}(i) {<} p(i) + c(i)\big) \\ \wedge \sum_{i \in I \cup J} \mathit{conf}(i) > 0 \ \wedge \ \gamma(\mathit{conf}) \text{ is defined}\} \end{aligned}$$

The set of configurations that do not lead to termination is defined as

$$\mathit{Conf1} = \{\mathit{conf} \in \mathit{Conf} \mid \sum_{i \in I} \mathit{conf}(i) > 0\}$$

and the set of all others is named $\mathit{Conf2} = \mathit{Conf} \setminus \mathit{Conf1}$. Now, for a given n we can check whether a_i^n conforms to a configuration as follows ($n \mid m$ represents the "n divides m" predicate):

$$\begin{aligned} \textbf{is\_conf}[\mathit{conf}, i](n) = \ (n = \mathit{conf}(i)) \vee \big(\mathit{conf}(i) > 0 \wedge c(i) > 0 \wedge n > p(i) \\ \wedge\, c(i) \mid (n - \mathit{conf}(i))\big) \end{aligned}$$

which says that n should either be the exact number specified in the configuration, or be greater than it by a multiple of $c(\mathsf{a}_i)$.

As one can expect, we need several list data types to deal with multi-party communications. In addition to the sorts *State* and *Nat* defined in Appendix C.1 we use the sorts *LState* and *LNat* to represent lists of natural numbers and states, respectively (cf. Appendix C.4). We also use the sort *ActPars* to represent different action parameter tuples that occur in the initial specification. Different actions may be parameterized by the same parameter sorts. In this case the values of the actual parameters have equal representations in the sort *ActPars*. The sorts E_i are used to represent the tuples of sorts that occur in the sum sequences of the equation (5.1) for X. These data types are tuple data types similar to *State*, with the exception that *ActPars* preserves type information for tuples. The sorts *LActPars* and LE_i represent lists of *ActPars* and E_i, respectively. All the list data types have the functions *len*, *cat* and *head*, representing the length of the list, concatenation of two list, and the first element of the list (undefined for the empty list), respectively. The following additional functions involving these data types are used in the definitions below:

$$
\begin{aligned}
&\mathit{is\_unique} : \mathit{LNat} \to \mathit{Bool}\\
&\mathit{is\_sorted} : \mathit{LNat} \to \mathit{Bool}\\
&\mathit{is\_each\_lower} : \mathit{LNat} \times \mathit{Nat} \to \mathit{Bool}\\
&\mathit{EQ} : \mathit{LActPars} \to \mathit{Bool}\\
&F_i : \mathit{LState} \times \mathit{LE}_i \to \mathit{LActPars}\\
&C_i : \mathit{LState} \times \mathit{LE}_i \to \mathit{Bool}
\end{aligned}
$$

The function *is_unique* checks if all list elements are unique, the function *is_sorted* checks if the list is sorted, and the function *is_each_lower* checks if each of the list elements is less than some natural number. The functions F_i model application of the terms $\overrightarrow{f_i}$ to each pair of elements in the argument lists, the functions C_i model conjunction of c_i applied to each pair of the elements, and the function *EQ* checks if all of the list elements are equal.

In addition to the data types *LM* and *ML* we use the sort *LLM* to represent lists of *LM*s (cf. Appendix C.5). The following additional functions involving this data type are used in the definitions below:

$\mathit{getfn} : \mathit{LM} \times \mathit{LNat} \to \mathit{LState}$	– get n first layer elements
$\mathit{replfn} : \mathit{LM} \times \mathit{LNat} \times \mathit{LLM} \to \mathit{LM}$	– replace n first layer elements with elements of *LLM*
$\mathit{remfn} : \mathit{LM} \times \mathit{LNat} \to \mathit{LM}$	– remove n first layer elements
$\mathit{mkllm}_i : \mathit{LState} \times \mathit{LE}_i \to \mathit{LLM}$	

The function mkllm_i applies the term $\mathbf{mklm}_i[p_i]$ to each pair of elements in the argument lists.

We use the following meta-symbols in the resulting LPE definition:

$$
\begin{aligned}
&\mathbf{cat}[l_0, \ldots, l_m] = \mathit{cat}(l_0, \ldots \mathit{cat}(l_{m-1}, l_m) \ldots)\\
&\mathbf{mkllm}[p_i](\mathit{ld}, \overrightarrow{\mathit{le}_i}) = \mathit{mkllm}_i(\mathit{ld}, \overrightarrow{\mathit{le}_i}) \text{ for } i \in I\\
&\mathbf{mkllm}[p_j](\mathit{ld}, \overrightarrow{\mathit{le}_j}) = \mathit{add}(\mathit{LM0}, \mathit{LLM0}) \text{ for } j \in J
\end{aligned}
$$

Assume the system G_7 consists of process equation X as defined in (5.1) with the sets of indices $J = \{0, \ldots, k\}$ and $I = \{k+1, \ldots, m\}$. We can now define a system L

consisting of process equation Z, which mimics behavior of X, in the following way:

$$
\begin{aligned}
&\mathsf{Z}(lm{:}LM) = \\
&\sum_{conf \in Conf1} \sum_{ln_0:LNat} \cdots \sum_{ln_m:LNat} \sum_{\overrightarrow{le_0:LE_0}} \cdots \sum_{\overrightarrow{le_m:LE_m}} \\
&\qquad \gamma(conf)(\overrightarrow{f_{mc}}(\overrightarrow{getf1(lm, head(ln))}, head(\overrightarrow{le_{mc}}))) \cdot \mathsf{Z}(replfn(lm, ln, \\
&\qquad\quad \mathbf{cat}[\mathbf{mkllm}[p_0](\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, \mathbf{mkllm}[p_m](\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})])) \\
&\qquad\quad \lhd\ ln \neq LNat0 \wedge len(ln) \leq lenf(lm) \wedge is\_unique(ln) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq i \leq m} is\_sorted(ln_i) \wedge \bigwedge\nolimits_{0 \leq i \leq m} is\_each\_lower(lenf(lm), ln_i) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq i \leq m} \mathbf{is\_conf}[conf, i](len(ln_i)) \wedge \bigwedge\nolimits_{0 \leq i \leq m} len(ln_i) = len(\overrightarrow{le_i}) \\
&\qquad\quad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})]) \\
&\qquad\quad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m}) \rhd \delta \\
&+ \sum_{conf \in Conf2} \sum_{ln_0:LNat} \cdots \sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}} \cdots \sum_{\overrightarrow{le_k:LE_k}} \\
&\qquad \gamma(conf)(\overrightarrow{f_{mc}}(\overrightarrow{getf1(lm, head(lnJ))}, head(\overrightarrow{le_{mc}}))) \cdot \mathsf{Z}(remfn(lm, lnJ)) \\
&\qquad\quad \lhd\ lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq j \leq k} is\_sorted(ln_j) \wedge \bigwedge\nolimits_{0 \leq j \leq k} is\_each\_lower(lenf(lm), ln_j) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq j \leq k} \mathbf{is\_conf}[conf, j](len(ln_j)) \wedge \bigwedge\nolimits_{0 \leq j \leq k} len(ln_j) = len(\overrightarrow{le_j}) \\
&\qquad\quad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})]) \\
&\qquad\quad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k}) \\
&\qquad\quad \wedge remfn(lm, lnJ) \neq \langle\rangle \rhd \delta \\
&+ \sum_{conf \in Conf2} \sum_{ln_0:LNat} \cdots \sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}} \cdots \sum_{\overrightarrow{le_k:LE_k}} \\
&\qquad \gamma(conf)(\overrightarrow{f_{mc}}(\overrightarrow{getf1(lm, head(lnJ))}, head(\overrightarrow{le_{mc}}))) \\
&\qquad\quad \lhd\ lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq j \leq k} is\_sorted(ln_j) \wedge \bigwedge\nolimits_{0 \leq j \leq k} is\_each\_lower(lenf(lm), ln_j) \\
&\qquad\quad \wedge \bigwedge\nolimits_{0 \leq j \leq k} \mathbf{is\_conf}[conf, j](len(ln_j)) \wedge \bigwedge\nolimits_{0 \leq j \leq k} len(ln_j) = len(\overrightarrow{le_j}) \\
&\qquad\quad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})]) \\
&\qquad\quad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k}) \\
&\qquad\quad \wedge remfn(lm, lnJ) = \langle\rangle \rhd \delta
\end{aligned}
$$

where $ln = \mathbf{cat}[ln_0, \ldots, ln_m]$, $lnJ = \mathbf{cat}[ln_0, \ldots, ln_k]$,
and $mc = min\{n \in Nat \mid conf(n) > 0\}$.

The first set of summands of the LPE represents the case when the process cannot terminate, because at least one of the communicating components is not terminating (for some $i \in I$ we have $conf(i) > 0$). The sum variables $ln_0, \ldots, ln_m$ represent lists of numbers of ready components that will communicate by performing actions $\mathsf{a}_0, \ldots, \mathsf{a}_m$ from the process equation for X, respectively. The condition of the summand makes sure that the total number of communicating components is not zero and not bigger than the total number of first layer elements. Moreover, the same component should not occur more than once, the order of the components is not important, and the numbers, the components are indexed by, are in range (smaller than $lenf(lm)$). Finally it is checked that the number of components performing each particular action conforms to the chosen configuration. The variables $\overrightarrow{le_0}, \ldots, \overrightarrow{le_m}$ represent lists of the sum parameter vectors $\overrightarrow{e_i}$ from the process equation for X. The length of each list should be equal to the number of components performing the corresponding action. We note that not all of the sums for $ln_0, \ldots, ln_m$ and $\overrightarrow{le_0}, \ldots, \overrightarrow{le_m}$ are needed for each configuration. For instance if in a particular configuration we have $conf(i) = 0$, then the sums for ln_i and $\overrightarrow{le_i}$ can be dropped. This is because the only valid representation of ln_i and $\overrightarrow{le_i}$ will be the empty list, and all other conjuncts of the condition involving them will be equal to true.

Furthermore, the other conditions necessary to make communication possible are: the initial conditions c_i are satisfied for all of the components, and the parameters of communicating actions are equal. We use the function $\overrightarrow{f_{mc}}$ applied to the first communicating component to get the values of the action parameters. To figure out what the next state of the process Z is, we replace the elements of the first layer of lm that took part in the communication with the next states these components would have in the process X ($LM0$ in case a particular component terminates).

The other two sets of summands represent the configurations that only involve the terminating actions of the equation for X. The difference between the two is in whether after this communication the lm becomes equal to $LM0$. If this is the case, then the LPE Z terminates, and otherwise it continues the execution.

The following theorem states the correctness of our construction.

Theorem 5.2.3. $(\mathsf{X}(\overrightarrow{d}), G_7) \overset{ind}{\Rightarrow} (\mathsf{Z}(seq1(\overrightarrow{d}, LM0)), L)$.

5.2.4 Multi-Party Communication with Renaming

For the case with the renaming operations we cannot use the communication configurations because we do not know to what action labels the initial action labels performed by the components will be renamed. That is why we have to expect that the resulting action can be any action to which one of the actions a_i can be renamed by a renaming function.

In addition to the data types ALM and AML we use the sort $LALM$ to represent lists of ALMs, the sort $LAct$ to represent lists of Acts and the sort $ActDT$ to represent either an action label, or τ, or δ (cf. Appendix C.6). The following additional functions involving these data types are used in the definitions of the resulting LPE:

$$\begin{aligned}
&is_act : Act \times LALM \times LNat \times LAct \to Bool\\
&is_tau : LALM \times LNat \times LAct \to Bool\\
&mklact : Nat \times Act \to LAct\\
&\overrightarrow{f0} : ALM \times LNAT \times \ldots \times LNAT \times E_0 \times \ldots \times E_n \to ActPars\\
&mkllm_i : LState \times LE_i \to LALM
\end{aligned}$$

The function *is_act* checks if a list of components can communicate by performing an action from the list, and the result of this communication is the given action. The function *is_tau* does the same, but checks that the result is τ. The function *mklact* generates the list of n actions a. The function $\overrightarrow{f0}$ can be defined as:

$$\begin{aligned}
&\overrightarrow{f0}(lm, ln_0, \ldots, ln_n, \overrightarrow{e_0}, \ldots, \overrightarrow{e_n})\\
&\qquad = \overrightarrow{f_l}(\overrightarrow{getf1d(lm, head(ln_l))}, \overrightarrow{e_l}) \text{ for } l = \mathbf{min}\{i \mid len(ln_i) > 0 \ \vee \ i = n\}
\end{aligned}$$

The meaning of this definition is that we find the number l of the first ready component taking part in the communication, and apply the corresponding function vector $\overrightarrow{f_l}$ to get the values of the action parameters. The function $mkllm_i$ applies the term $\mathbf{mklm}_i[p_i]$ to each pair of elements in the argument lists.

Assume the system G_7 consists of process equation X as defined in (5.4). A system L consisting of process equation Z, which mimics behavior of X, is defined in Appendix A.2. The correctness statement is similar to the case with handshaking:

Theorem 5.2.4. $(\mathsf{X}(\overrightarrow{d}), G_7) \overset{ind}{\Rightarrow} (\mathsf{Z}(seq1(Ann0, \overrightarrow{d}, LM0)), L)$.

Summarizing Section 5.2 and the entire transformation, for any X^s from the initial μCRL specification we have

$$(\mathsf{X}^s(\overrightarrow{t}), G) \overset{ind}{\Rightarrow} (\mathsf{Z}(seq1(Ann0, (s, M_{\mathsf{X}^s}(\overrightarrow{t})), LM0)), L)$$

and the current specification contains definitions of the data types from Appendix C (for the data type dependencies we refer to Figure C.1 in that Appendix).

Chapter 6

Linearization in Timed μCRL

6.1 Algebraic Theory of Timed μCRL

6.1.1 Syntax and Axioms of Timed μCRL

First we define the equational theory of the sort *Time* by defining its signature and the axioms. Many of the axioms are taken from, or inspired by [47, 60].

Definition 6.1.1. The signature of *Time* consists of

- constant $\mathbf{0}$,
- function $leq : Time \times Time \to Bool$, which we often abbreviate to $\leq$;
- function $if : Bool \times Time \times Time \to Time$.
- function $eq : Time \times Time \to Bool$, which we often abbreviate to $=$;
- function $max : Time \times Time \to Time$;
- function $min : Time \times Time \to Time$.

The time domain is a totally ordered domain (ordered by $\leq$) with zero element $\mathbf{0}$ and *min* and *max* operations which form a distributive lattice.

Definition 6.1.2. The axioms of *Time* are the ones presented in Table 6.1.

A number of identities of sort *Time* are provable from the axioms (cf. Lemma B.2.1 in Appendix B.2.1). According to the axioms, every boolean term can be transformed to an equivalent one that does not contain *if*, but only boolean connectives and terms of the form $t \leq u$, where t is a variable of sort *Time* and u is either a variable of sort *Time* or $\mathbf{0}$. Every term of sort *Time* is either $\mathbf{0}$, a variable, or is of the form $if(b, t, u)$ for t and u other terms of sort *Time*. The above mentioned form has two extremes: one where all boolean terms b are variables, and the other is where every variable of sort time (and $\mathbf{0}$) occurs at most once.

$t \leq u \wedge u \leq w \approx t \leq u \wedge u \leq w \wedge t \leq w$	(Time1$'$)
$t \leq u \wedge \neg\, w \leq u \approx t \leq u \wedge \neg\, w \leq u \wedge \neg\, w \leq t$	(Time1$''$)
$\neg\, u \leq t \wedge u \leq w \approx \neg\, u \leq t \wedge u \leq w \wedge \neg\, w \leq t$	(Time1$'''$)
$\mathbf{0} \leq t \approx \mathbf{t}$	(Time2)
$t \leq u \vee u \leq t \approx \mathbf{t}$	(Time3)
$eq(t,u) \approx t \leq u \wedge u \leq t$	(Time5)
$min(t,u) \approx if(t \leq u, t, u)$	(Time6)
$max(t,u) \approx if(u \leq t, t, u)$	(Time6$'$)
$if(\mathbf{t}, t, u) \approx t$	(Time7)
$if(\neg b, t, u) \approx if(b, u, t)$	(Time8$'$)
$if(b_1 \vee b_2, t, u) \approx if(b_1, t, if(b_2, t, u))$	(Time9)
$if(b_1, if(b_2, t, u), w) \approx if(b_1 \wedge b_2, t, if(b_1, u, w))$	(Time10)
$if(t \leq u \wedge u \leq t, t, u) \approx u$	(Time11)
$t \leq if(b, u, w) \approx (b \wedge t \leq u) \vee (\neg b \wedge t \leq w)$	(Time12)
$if(b, u, w) \leq t \approx (b \wedge u \leq t) \vee (\neg b \wedge w \leq t)$	(Time13)

Table 6.1: Basic axioms of *Time*.

The latter form is useful for proving time identities in the following way: if we order the time variables occurring in a term as $\mathbf{0} < t_1 < \ldots < t_n$, then with the help of the axioms we can transform every term to the form

$$if(b_1, t_{i_0}, if(b_2, t_{i_1}, \ldots if(b_n, t_{i_{m-1}}, t_{i_m}) \ldots))$$

with indices such that $t_{i_k} < t_{i_{k+1}}$. Moreover, the conditions $b_1, \ldots, b_n$ can be made pairwise distinct, i.e. having the property that $i \neq j \rightarrow b_i \wedge b_j \approx \mathbf{f}$ (cf. Lemma B.2.1.17). In addition, the conditions $b_1, \ldots, b_n$ can be made such that if $eq(t_{i_k}, t_{i_{k+1}}) \approx \mathbf{t}$, then $b_k \approx \mathbf{t}$. (see Lemma B.2.1.19). This gives us a method for proving identities of sort *Time*.

Next we define the binding-equational theory of timed μCRL by defining its signature and the axioms. Many of the axioms are taken from, or inspired by [55, 49].

Definition 6.1.3 (Signature of Timed μCRL). The signature of timed μCRL consists of data sorts (or 'data types') including *Bool* and *Time* as defined above, and a distinct sort *Proc* of processes. Each data sort D is assumed to be equipped with a binary function $eq : D \times D \rightarrow Bool$. (This requirement can be weakened by demanding such functions only for data sorts that are parameters of communicating actions). The operational signature of timed μCRL is parameterized by the finite set of action labels *ActLab* and a partial commutative and associative function γ : $ActLab \times ActLab \rightarrow ActLab$ such that $\gamma(\mathsf{a}_1, \mathsf{a}_2) \in ActLab$ implies that $\mathsf{a}_1, \mathsf{a}_2$ and $\gamma(\mathsf{a}_1, \mathsf{a}_2)$ have parameters of the same sorts. The process operations are the ones listed below:

- actions $\mathsf{a}(\overrightarrow{t})$ parameterized by data terms $\overrightarrow{t}$, where $\mathsf{a} \in ActLab$ is an action label. More precisely, a is an operation $\mathsf{a} : \overrightarrow{D_{\mathsf{a}}} \rightarrow Proc$. We write $type(\mathsf{a})$ for $\overrightarrow{D_{\mathsf{a}}}$.

- constants δ and τ of sort *Proc*.
- binary operations $+, \cdot, \|, \lfloor\!\lfloor, |, \ll$ defined on *Proc*, where $|$ is defined using γ.
- unary *Proc* operations $\partial_H, \tau_I, \rho_R, \partial\mathcal{U}$ for each set of action labels $H, I \subseteq \mathit{ActLab}$ and an action label renaming function $R : \mathit{ActLab} \to \mathit{ActLab}$ such that a and $R(\mathsf{a})$ have parameters of the same sorts. Such functions R we call *well-defined* action label renaming functions.
- a ternary operation $\_ \lhd \_ \rhd \_ : \mathit{Proc} \times \mathit{Bool} \times \mathit{Proc} \to \mathit{Proc}$.
- binders $\sum_{d:D}$ defined on *Proc*, for each data variable d of sort D.
- binary operations ${}^{\varsigma} : \mathit{Proc} \times \mathit{Time} \to \mathit{Proc}$, and $\gg$ and $\ggg$: $\mathit{Time} \times \mathit{Proc} \to \mathit{Proc}$.

A key feature of timed μCRL is that it can be expressed at which time a certain action must take place. This is done using the *"at"-operator*. The process $p^{\varsigma}t$ behaves like the process p, with the restriction that the first action of p must start at time t. All time values in timed μCRL represent *absolute* time, i.e. the time that has passed since the system was initialized, and not the time since the previous action was performed. For a thorough treatment of timing mechanisms in process algebra, and the relations among these mechanisms, we refer to [9]. For a way to interpret timed automata [3] in timed μCRL we refer to [101, Chapter 6].

Another key feature of timed μCRL is that it can be expressed that a process can delay till a certain time. The process $p + \delta {}^{\varsigma} t$ can certainly delay till time t, but can possibly delay longer, depending on p. Consequently, the process $\delta^{\varsigma}\mathbf{0}$ can neither delay nor perform actions, and the process δ can delay for an arbitrary long time, but cannot perform any action. We follow the intuition that a process that can delay till time t can also delay till an earlier moment, and a process that can perform a *first* action at time t can also delay till time t.

The process $p {}^{\varsigma} t$ can delay till at most time t. If p consists of several alternatives, then only those with the first actions starting at time t will remain in $p {}^{\varsigma} t$. The alternatives that start earlier than t will express that $p {}^{\varsigma} t$ can delay till that earlier time. The alternatives that start later than t will express that $p {}^{\varsigma} t$ can wait till time t (but not till that later time).

The *ultimate delay* operator $\partial\mathcal{U}(p)$ expresses the process, which can delay as long as p can, but cannot perform any action. The *initialization* operator $t \gg p$ expresses the process in which all alternatives of p that start earlier than t are left out, but an alternative to delay till time t is added. The *weak initialization* operator $t \ggg p$ expresses the process in which all alternatives of p that start earlier than t are replaced by the ability to delay till those earlier times. Thus the process $t \ggg p$ can delay till the same time as p, while $t \gg p$ can delay till at least time t, which can be longer than p could delay. The *before* operator $p \ll q$ expresses the process in which all alternatives of p that start later than $\partial\mathcal{U}(q)$ are replaced by the abilities to delay till $\partial\mathcal{U}(q)$. Thus $p \ll q$ cannot delay longer than both p and q. The ultimate delay $\partial\mathcal{U}(p)$ of process p can be expressed in terms of $\ll$ as $\delta \ll p$. This process cannot perform actions and can delay as long as p could (because δ can delay till any time).

Definition 6.1.4 (Axioms of Timed μCRL). Axioms of timed μCRL are the ones presented in Tables B.1,B.2,B.3,B.4,B.5,B.6, and B.7 in Appendix B.1 and Table 2.8 in Chapter 2. We assume that

- the descending order of binding strength of operators is: $^{\varsigma}$, $\cdot$, $\{\gg, \ggg, \ll\}$, $\{\|, \lfloor\!\lfloor, |\}$, $\lhd\rhd$, $\sum$, $+$;
- x, y, z are variables of sort *Proc*;
- c, c_1, c_2 are variables of sort *Bool*;
- $d, d^1, d^n, d', \ldots$ are data variables (but d in $\sum_{d:D}$ is not a variable);
- b stands for either $\mathsf{a}(\overrightarrow{d})$, or τ, or δ;
- $\overrightarrow{d} = \overrightarrow{d'}$ is an abbreviation for $eq(d^1, d'^1) \wedge \cdots \wedge eq(d^n, d'^n)$, where $\overrightarrow{d} = d^1, \ldots, d^n$ and $\overrightarrow{d'} = d'^1, \ldots, d'^n$;
- the axioms where p and q occur are schemata ranging over all terms p and q of sort *Proc*, *including* those in which d occurs freely;
- the axiom (SUM2) is a scheme ranging over all terms r of sort *Proc* in which d *does not* occur freely;
- t, u, w are variables of sort *Time*.

To prove identities in timed μCRL we use a combined many-sorted calculus, which for the sort of processes has the rules of binding-equational calculus, for the sorts of booleans and time has the rules of equational calculus, while other data sorts may include induction principles which could be used to derive process identities as well. We note that the derivation rules of binding-equational calculus do not allow to substitute terms containing free variables if they become bound. For example, in axiom (SUM1) we cannot substitute $\mathsf{a}(d)$ for x.

Definition 6.1.5. Two process terms p_1 and p_2 are *(unconditionally) timed equivalent* (notation $p_1 \approx p_2$) if $p_1 \approx p_2$ is derivable from the axioms of timed μCRL and boolean and time identities by using many sorted binding-equational calculus. In this case we write $\{\mu\text{CRL}, \mathit{BOOL}, \mathit{TIME}\} \vdash_{eBA} p_1 \approx p_2$. Here *BOOL* and *TIME* is used to refer to the specification of the booleans and time, respectively, and the use of equational logic for deriving boolean and time identities.

Two process terms p_1 and p_2 are *conditionally timed equivalent* if

$$\{\mu\text{CRL}, \mathit{BOOL}, \mathit{TIME}, \mathit{DATA}\} \vdash_{eBA} p_1 \approx p_2.$$

Here *DATA* is used to refer to the specification of all data sorts involved, and all proof rules that may be applied.

Similar to the μCRL axiomatization in Chapter 2, a number of identities that can be found as axioms of timed μCRL, are derivable in our setting, but nevertheless, we shall still call them axioms of timed μCRL.

Lemma 6.1.6. *The following identities are derivable from the axioms of timed μCRL.*

$$b \mid (b' \cdot x) \approx (b \mid b') \cdot x \qquad \text{(CM6)}$$
$$x \mid (y + z) \approx x \mid y + x \mid z \qquad \text{(CM9)}$$
$$b \mid \tau \approx \delta \qquad \text{(CT2)}$$
$$\delta \mid b \approx \delta \qquad \text{(CD1)}$$
$$b \mid \delta \approx \delta \qquad \text{(CD2)}$$
$$x \mid (y \lhd c \rhd \delta) \approx (x \mid y) \lhd c \rhd \delta \qquad \text{(Cond9')}$$
$$\sum_{d:D}(x \mid p) \approx x \mid (\sum_{d:D} p) \qquad \text{(SUM7')}$$

Proof. The axiom (CD2) is a special instance of (SCDT2), and the rest are derivable from symmetric axioms using (SC3). □

A number of other identities derivable from the axioms of timed μCRL, booleans and time are presented in Lemma B.2.3 and B.2.4 in Appendix B.2.2. We also note that the identities derivable in Section 2.2 are also derivable in the timed μCRL setting, as well as the equivalence theory presented in Section 3.3.

6.1.2 Timed μCRL Specifications

For the purpose of this chapter we restrict to timed μCRL specifications that do not contain left merge ($\mathbin{|\!\lfloor}$), communication ($|$), ultimate delay ($\partial\mathcal{U}$), and before ($\ll$) operators explicitly. These operators were introduced to allow the finite axiomatization of parallel composition ($\|$) and timing constructs in the bisimulation setting, and they are hardly used explicitly in timed μCRL specifications.

We consider systems of process equations with the right-hand sides from the following subset of timed μCRL terms:

$$p ::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p{+}p \mid p{\cdot}p \mid p\|p \mid \sum_{d:D} p \mid p{\lhd}c{\rhd}p \mid \partial_H(p) \mid \tau_I(p) \mid \rho_R(p)$$
$$\mid p \,{}^{\varsigma}\, t \mid t \gg p \mid t \ggg p \qquad (6.1)$$

All the requirements set for μCRL specifications in Section 4.1 are still valid for the timed μCRL, and the linearization problem is formulated in a similar way. The problem of linearization of a timed μCRL specification defined by $(\mathsf{X}(\overrightarrow{t}), G)$ consists of the generation of a new timed μCRL specification which

- depends on the same set of actions and communication function,
- contains all data definitions of the original one, and, possibly, definitions of the auxiliary data types,
- is defined by $(\mathsf{Z}(\mathrm{m}_{\mathsf{X}}(\overrightarrow{t})), L)$, where L contains exactly one process equation for Z in linear form (defined later), and m_{X} is a mapping from $pars(\mathsf{X}, G)$ to $pars(\mathsf{Z}, L)$,

such that $(\mathsf{X}(\overrightarrow{t}), G) \overset{ind}{\Rightarrow} (\mathsf{Z}(\mathrm{m}_{\mathsf{X}}(\overrightarrow{t})), L)$.

6.2 Transformation to Post-TPEGNF

As input for the linearization procedure we take a timed μCRL process definition $(\mathsf{X}(\overrightarrow{t}), G)$ such that PNUDG of G is acyclic. In this section we transform G into a system of process equations G_4 in post-Timed Parallel Extended Greibach Normal Form. The resulting system will contain process equations for all process names in $|G|$ with the same names and types of data parameters involved, as well as, possibly, other process equations.

6.2.1 Normal Forms

Below we define several normal forms for systems of process equations in timed μCRL, similar to Timed Parallel Extended Greibach Normal Form (TPEGNF). A system is said to be in one of these forms if all of its equations are in the respective form.

From this point on we assume that $\mathsf{a}(\overrightarrow{t})$ with possible indices can also be an abbreviation for τ. This is done to make the normal form representations more concise.

Definition 6.2.1. A timed μCRL process equation is in *pre-TPEGNF* if it is of the form:

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0}$$

where $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$\begin{aligned} p ::= {} & \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p\cdot p \mid p \parallel p \mid \rho_R(\tau_I(\partial_H(p\parallel p))) \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \\ & \mid \mathsf{a}(\overrightarrow{t}){}^{\varsigma} t \mid \delta {}^{\varsigma} t \mid \mathsf{Y}(\overrightarrow{t}){}^{\varsigma} t \mid (p\parallel p){}^{\varsigma} t \mid \rho_R(\tau_I(\partial_H((p\parallel p){}^{\varsigma} t))) \\ & \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}){}^{\varsigma} t))) \end{aligned} \tag{6.2}$$

A timed μCRL process equation is in *TPEGNF* iff it is of the form:

$$\begin{aligned} \mathsf{X}(\overrightarrow{d{:}D}) = {} & \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) {}^{\varsigma} t_i(\overrightarrow{d,e_i}) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0} \\ & + \sum_{j\in J} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) {}^{\varsigma} t_j(\overrightarrow{d,e_j}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta{}^{\varsigma}\mathbf{0} \\ & + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta {}^{\varsigma} t_\delta(\overrightarrow{d,e_\delta}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0} \end{aligned}$$

where I and J are disjoint, and all $p_i(\overrightarrow{d,e_i})$ have the following syntax:

$$\begin{aligned} p ::= {} & \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p\cdot p \mid p \parallel p \mid \rho_R(\tau_I(\partial_H(p\parallel p))) \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \\ & \mid p {}^{\varsigma} t \mid t \gg p \mid t \ggg p \end{aligned} \tag{6.3}$$

A timed μCRL process equation is in *refined-TPEGNF* iff it is of the same form as above, but the terms $p_i(\overrightarrow{d, e_i})$ have the following restricted syntax:

$$\begin{aligned} p &::= p \cdot p \mid p \parallel p \mid \rho_R(\tau_I(\partial_H(p \parallel p))) \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \mid p_1 \\ p_1 &::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p_1 {}^{\varsigma}\, t \mid t \gg p_1 \mid t \ggg p_1 \end{aligned} \tag{6.4}$$

A timed μCRL process equation is in *post-TPEGNF* iff it is of the same form as above, but the terms $p_i(\overrightarrow{d, e_i})$ have the following even more restricted syntax:

$$p ::= \mathsf{Y}(\overrightarrow{t}) \mid p \cdot p \mid p \parallel p \mid \rho_R(\tau_I(\partial_H(p \parallel p))) \mid \rho_R(\tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t})))) \tag{6.5}$$

A timed μCRL process equation is called *Timed Linear Process Equation (TLPE)* iff it is of the same form as above, but the terms $p_i(\overrightarrow{d, e_i})$ are recursive calls of the form $\mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d, e_i}))$ for some function vectors $\overrightarrow{g_i}$.

In the remainder of this Section we transform the initial system G into post-TPEGNF. First we apply a preprocessing step described in Section 4.2.1 that renames variables that are bound twice. Subsections 6.2.2 and 6.2.3 explain the transformation to pre-TPEGNF, Subsection 6.2.4 explains the transformation from pre-TPEGNF to TPEGNF, and Subsection 6.2.5 explains the transformation to post-TPEGNF through refined-TPEGNF.

6.2.2 Reduction by Simple Rewriting

By applying term rewriting we get an equivalent set of process equations to the given one, but with terms in right-hand sides having the more restricted form as presented in Table 6.2. This syntax is obtained by analyzing what kind of terms may occur in scope of which operation symbols. For example if δ occurs on the left of a sequential composition ($\delta{\cdot}x$), then the term can be reduced by rewriting it to δ. Such a restriction on the syntax makes the further transformation to pre-TPEGNF simpler.

The rewrite rules that we apply to the right-hand sides of the equations are listed in Tables 6.3, 6.4 and 6.5. The symbols $\sum_{d:D}$ are treated in this rewrite system as function symbols, not as binders. This is justified by the fact that we have renamed all nested bound variables, which allows the use of first order term rewriting. The mapping induced by the rewrite rules for a given system of process equations G is called $\mathit{rewr} : \mathit{Terms}(|G|) \to \mathit{Terms}(|G|)$.

Before applying rewriting we eliminate all terms of the form $\_ \lhd \_ \rhd \_$ with the third argument different from $\delta{}^{\varsigma}\mathbf{0}$, with the following rule:

$$y \not\equiv \delta{}^{\varsigma}\mathbf{0} \implies x \lhd c \rhd y \to x \lhd c \rhd \delta{}^{\varsigma}\mathbf{0} + y \lhd \neg c \rhd \delta{}^{\varsigma}\mathbf{0} \tag{RCOND3T}$$

We also eliminate all occurrences of $\gg$ and $\ggg$ with the following rules:

$$t \gg x \to \sum_{u:\mathit{Time}} x {}^{\varsigma}\, u \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \delta {}^{\varsigma}\, t \tag{RATBD0}$$

$$t \ggg x \to \sum_{u:\mathit{Time}} x {}^{\varsigma}\, u \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} \tag{RATD0}$$

Rewriting is performed modulo the identities presented in Table 6.6

$$
\begin{aligned}
p &::= \mathsf{a}(\vec{t}) \mid \delta \mid \mathsf{Y}(\vec{t}) \mid p+p \mid p_1 \cdot p \mid p_2 \parallel p_2 \mid \sum_{d:D} p_3 \mid p_4 \lhd c \rhd \delta^{\varsigma}\mathbf{0} \mid p^* \\
p_1 &::= \mathsf{a}(\vec{t}) \mid \mathsf{Y}(\vec{t}) \mid p+p \mid p_1 \cdot p \mid p_2 \parallel p_2 \mid \partial_H(p_5) \mid \tau_I(p_6) \mid \rho_R(p_7) \mid p_9 {}^{\varsigma} t \\
p_2 &::= \mathsf{a}(\vec{t}) \mid \mathsf{Y}(\vec{t}) \mid p+p \mid p_1 \cdot p \mid p_2 \parallel p_2 \mid \sum_{d:D} p_3 \mid p_4 \lhd c \rhd \delta^{\varsigma}\mathbf{0} \mid p^* \\
p_3 &::= \mathsf{a}(\vec{t}) \mid \mathsf{Y}(\vec{t}) \mid p_1 \cdot p \mid p_2 \parallel p_2 \mid \sum_{d:D} p_3 \mid p_4 \lhd c \rhd \delta^{\varsigma}\mathbf{0} \mid p^* \\
p_4 &::= \mathsf{a}(\vec{t}) \mid \delta \mid \mathsf{Y}(\vec{t}) \mid p_1 \cdot p \mid p_2 \parallel p_2 \mid p^* \\
p^* &::= \partial_H(p_5) \mid \tau_I(p_6) \mid \rho_R(p_7) \mid p_8 {}^{\varsigma} t \\
p_5 &::= \mathsf{Y}(\vec{t}) \mid p_2 \parallel p_2 \mid p_5 {}^{\varsigma} t \\
p_6 &::= p_5 \mid \partial_H(p_5) \\
p_7 &::= p_6 \mid \tau_I(p_6) \\
p_8 &::= \delta \mid p_9 \\
p_9 &::= \mathsf{a}(\vec{t}) \mid p_5
\end{aligned}
$$

Table 6.2: Syntax of terms after simple rewriting.

$$
\begin{aligned}
\delta \cdot x &\to \delta && \text{(RA7)} \\
\delta {}^{\varsigma} t \cdot x &\to \delta {}^{\varsigma} t && \text{(RA7T)} \\
x \parallel \delta &\to x \cdot \delta && \text{(RSCD1)} \\
(x \cdot \delta) \parallel y &\to (x \parallel y) \cdot \delta && \text{(RSCD2)} \\
\Big(\sum_{d:D} x\Big) \cdot y &\to \sum_{d:D} (x \cdot y) && \text{(RSUM5)} \\
(x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) \cdot y &\to (x \cdot y) \lhd c \rhd \delta^{\varsigma}\mathbf{0} && \text{(RCOND6T)} \\
\sum_{d:D} \delta &\to \delta && \text{(RSUM1}'\text{)} \\
\sum_{d:D} (x+y) &\to \sum_{d:D} x + \sum_{d:D} y && \text{(RSUM4)} \\
(x+y) \lhd c \rhd \delta^{\varsigma}\mathbf{0} &\to x \lhd c \rhd \delta^{\varsigma}\mathbf{0} + y \lhd c \rhd \delta^{\varsigma}\mathbf{0} && \text{(RCOND7T)} \\
\Big(\sum_{d:D} x\Big) \lhd c \rhd \delta^{\varsigma}\mathbf{0} &\to \sum_{d:D} x \lhd c \rhd \delta^{\varsigma}\mathbf{0} && \text{(RSUM12T)} \\
(x \lhd c_1 \rhd \delta^{\varsigma}\mathbf{0}) \lhd c_2 \rhd \delta^{\varsigma}\mathbf{0} &\to x \lhd c_1 \wedge c_2 \rhd \delta^{\varsigma}\mathbf{0} && \text{(RCOND4T)}
\end{aligned}
$$

Table 6.3: Rewrite rules defining *rewr* (Part 1).

Proposition 6.2.2. *The commutative/associative term rewriting system of Tables 6.3, 6.4 and 6.5 is terminating.*

Proof. Termination can be proved using the AC-RPO technique [90] for following order on the operations:

$$
{}^{\varsigma}t > \partial_H > \tau_I > \rho_R > \parallel > \cdot > \_ \lhd c \rhd \delta^{\varsigma}\mathbf{0} > \sum > + > \mathsf{a}(\vec{t}) > \delta
$$

□

$\partial_H(\mathsf{a}(\overrightarrow{t})) \to \delta$	if $\mathsf{a} \in H$	(RD2)
$\partial_H(\mathsf{a}(\overrightarrow{t})) \to \mathsf{a}(\overrightarrow{t})$	if $\mathsf{a} \notin H$	(RD1)
$\partial_H(\mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t) \to \delta \,{}^{c}\, t$	if $\mathsf{a} \in H$	(RD2T)
$\partial_H(\mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t) \to \mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t$	if $\mathsf{a} \notin H$	(RD1T)
$\partial_H(\tau) \to \tau$		(RD1′)
$\partial_H(\tau \,{}^{c}\, t) \to \tau \,{}^{c}\, t$		(RD1T′)
$\partial_H(\delta) \to \delta$		(RD2′)
$\partial_H(\delta \,{}^{c}\, t) \to \delta \,{}^{c}\, t$		(RD2T′)
$\partial_H(x + y) \to \partial_H(x) + \partial_H(y)$		(RD3)
$\partial_H(x \cdot y) \to \partial_H(x) \cdot \partial_H(y)$		(RD4)
$\partial_H\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \partial_H(x)$		(RSUM8)
$\partial_H(x \lhd c \rhd \delta \,{}^{c}\mathbf{0}) \to \partial_H(x) \lhd c \rhd \delta \,{}^{c}\mathbf{0}$		(RD5T)
$\partial_{H_1}(\partial_{H_2}(x)) \to \partial_{H_1 \cup H_2}(x)$		(RDD)
$\partial_H(\tau_I(x)) \to \tau_I(\partial_{H \setminus I}(x))$		(RDT)
$\partial_H(\rho_R(x)) \to \rho_R(\partial_{R^{-1}(H)}(x))$		(RDR)
$\tau_I(\mathsf{a}(\overrightarrow{t})) \to \tau$	if $\mathsf{a} \in I$	(RT2)
$\tau_I(\mathsf{a}(\overrightarrow{t})) \to \mathsf{a}(\overrightarrow{t})$	if $\mathsf{a} \notin I$	(RT1)
$\tau_I(\mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t) \to \tau \,{}^{c}\, t$	if $\mathsf{a} \in I$	(RT2T)
$\tau_I(\mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t) \to \mathsf{a}(\overrightarrow{t}) \,{}^{c}\, t$	if $\mathsf{a} \notin I$	(RT1T)
$\tau_I(\tau) \to \tau$		(RT2′)
$\tau_I(\tau \,{}^{c}\, t) \to \tau \,{}^{c}\, t$		(RT2T′)
$\tau_I(\delta) \to \delta$		(RT1′)
$\tau_I(\delta \,{}^{c}\, t) \to \delta \,{}^{c}\, t$		(RT1T′)
$\tau_I(x + y) \to \tau_I(x) + \tau_I(y)$		(RT3)
$\tau_I(x \cdot y) \to \tau_I(x) \cdot \tau_I(y)$		(RT4)
$\tau_I\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \tau_I(x)$		(RSUM9)
$\tau_I(x \lhd c \rhd \delta \,{}^{c}\mathbf{0}) \to \tau_I(x) \lhd c \rhd \delta \,{}^{c}\mathbf{0}$		(RT5T)
$\tau_{I_1}(\tau_{I_2}(x)) \to \tau_{I_1 \cup I_2}(x)$		(RTT)
$\tau_I(\rho_R(x)) \to \rho_R(\tau_{R^{-1}(I)}(x))$		(RTR)

Table 6.4: Rewrite rules defining *rewr* (Part 2).

Lemma 6.2.3. *For any process term p not containing $\gg, \ggg$ operations, and not containing $p_1 \lhd c \rhd p_2$, where $p_2 \not\equiv \delta$, we have that $rewr(p)$ has the syntax defined in Table 6.2.*

Proof. Let $q = rewr(p)$. It can be seen from the rewrite rules that they preserve the syntax (6.1). Suppose q does not satisfy the syntax defined in Table 6.2. All of the possibilities for q that exist imply that q is reducible by one of the rules in Tables 6.4 and 6.5. □

$\rho_R(\mathsf{a}(\overrightarrow{t})) \to R(\mathsf{a})(\overrightarrow{t})$	(RR1)
$\rho_R(\mathsf{a}(\overrightarrow{t}) ^{\varsigma} t) \to R(\mathsf{a})(\overrightarrow{t}) ^{\varsigma} t$	(RR1T)
$\rho_R(\tau) \to \tau$	(RRT)
$\rho_R(\tau ^{\varsigma} t) \to \tau ^{\varsigma} t$	(RRTT)
$\rho_R(\delta) \to \delta$	(RRD)
$\rho_R(\delta ^{\varsigma} t) \to \delta ^{\varsigma} t$	(RRDT)
$\rho_R(x+y) \to \rho_R(x) + \rho_R(y)$	(RR3)
$\rho_R(x \cdot y) \to \rho_R(x) \cdot \rho_R(y)$	(RR4)
$\rho_R\Big(\sum_{d:D} x\Big) \to \sum_{d:D} \rho_R(x)$	(RSUM10)
$\rho_R(x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) \to \rho_R(x) \lhd c \rhd \delta^{\varsigma}\mathbf{0}$	(RR5T)
$\rho_{R_1}(\rho_{R_2}(x)) \to \rho_{R_1 \circ R_2}(x)$	(RRR)
$(x+y) ^{\varsigma} t \to x ^{\varsigma} t + y ^{\varsigma} t$	(RATA2)
$(x \cdot y) ^{\varsigma} t \to x ^{\varsigma} t \cdot y$	(RATA3)
$\Big(\sum_{d:D} x\Big) ^{\varsigma} t \to \sum_{d:D} x ^{\varsigma} t$	(RATA4)
$(x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) ^{\varsigma} t \to x ^{\varsigma} t \lhd c \rhd \delta^{\varsigma}\mathbf{0}$	(RATA5)
$(\partial_H(x)) ^{\varsigma} t \to \partial_H(x ^{\varsigma} t)$	(RD7)
$(\tau_I(x)) ^{\varsigma} t \to \tau_I(x ^{\varsigma} t)$	(RT7)
$(\rho_R(x)) ^{\varsigma} t \to \rho_R(x ^{\varsigma} t)$	(RR7)
$(\mathsf{a}(\overrightarrow{t}) ^{\varsigma} t) ^{\varsigma} u \to \mathsf{a}(\overrightarrow{t}) ^{\varsigma} t \lhd t = u \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma} min(t,u)$	(RATA1)
$(\tau ^{\varsigma} t) ^{\varsigma} u \to \tau ^{\varsigma} t \lhd t = u \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma} min(t,u)$	(RATAT1)
$(\delta ^{\varsigma} t) ^{\varsigma} u \to \delta ^{\varsigma} min(t,u)$	(RATAD1)

Table 6.5: Rewrite rules defining *rewr* (Part 3).

$$x + y \approx y + x$$
$$x + (y + z) \approx (x + y) + z$$
$$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$$
$$x \parallel y \approx y \parallel x$$
$$x \parallel (y \parallel z) \approx (x \parallel y) \parallel z$$

Table 6.6: The rewriting is performed modulo these identities.

Proposition 6.2.4. *Let G_2 be the result of applying the rewriting to G_1. Then $G_2 = G_1$.*

Proof. Taking into account that G_1 does not contain nested occurrences of bound variables, each rewrite rule is a consequence of the axioms of timed μCRL. By Lemma 3.5.1 we get $G_2 = G_1$. □

As a result of applying simple rewriting the number of equations obviously remains the same. The number of occurrences of action labels and process names does not increase during rewriting.

6.2.3 Adding New Process Equations

This step is similar to the one described in Sections 4.2.3 and 5.1.2. The di the step described in the latter Section (for the case of full μCRL) is in th that is treated similarly to sequential composition. We extend the tra S_1 and S_2 described in Sections 5.1.2 with the following rules:

$$S_1(S, p \mathbin{{}^{\varsigma}} t) \to S_2(S, p \mathbin{{}^{\varsigma}} t) \qquad S_2(S, p \mathbin{{}^{\varsigma}} t) \to S_2(S,p) \mathbin{{}^{\varsigma}} t$$

and we replace the two condition rules to handle $\delta \mathbin{{}^{\varsigma}} \mathbf{0}$ in place of δ. As replacing every equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = p_\mathsf{X}$ of G_2 by $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = S_1(\overrightarrow{d_\mathsf{X}{:}D}$ a system of process equations G_3, which is in pre-PEGNF.

Lemma 6.2.5. *All process equations in G_3 are in pre-TPEGNF.*

Proof. Similar to Lemma 4.2.7.

Proposition 6.2.6. *For any process name X in G_2 we have* $(\mathsf{X}, G_3) =$

Proof. Similar to Proposition 4.2.8.

The transformation described in this subsection does not increase the The number of process equations may increase linearly in the size of original system.

6.2.4 Guarding

Next we transform the equations of G_3 to TPEGNF. To this end, we use *guard* : *DVar* $\times$ *Terms*$(|G|) \to$ *Terms*$(|G|)$, which replaces unguarded o process names with the right-hand sides of their defining equations. It follows:

$$\mathit{guard}\Big(S, \sum_{i \in I} \sum_{\overrightarrow{e_i:E_i}} p_i \lhd c_i \rhd \delta \mathbin{{}^{\varsigma}} \mathbf{0}\Big) = \mathit{rewr}\Big(\sum_{i \in I} \sum_{\overrightarrow{e_i:E_i}} \mathit{guard}(S \cup \{\overrightarrow{e_i}\}, p_i) \lhd$$

$$\mathit{guard}(S, \mathsf{a}(\overrightarrow{t})) = \sum_{u:\mathit{Time}} \mathsf{a}(\overrightarrow{t}) \mathbin{{}^{\varsigma}} u \qquad \text{where } u \text{ is a fresh variable } (u \notin$$

$$\mathit{guard}(S, \mathsf{a}(\overrightarrow{t}) \mathbin{{}^{\varsigma}} t) = \mathsf{a}(\overrightarrow{t}) \mathbin{{}^{\varsigma}} t$$

$$\mathit{guard}(S, \delta) = \sum_{u:\mathit{Time}} \delta \mathbin{{}^{\varsigma}} u \qquad \text{where } u \text{ is a fresh variable } (u \notin S)$$

$$\mathit{guard}(S, \delta \mathbin{{}^{\varsigma}} t) = \delta \mathbin{{}^{\varsigma}} t$$

$$\mathit{guard}(S, \mathsf{Y}(\overrightarrow{t})) = \mathit{guard}\Big(S, S_0(S \setminus \{\mathit{pars}(\mathsf{Y})\}, \mathit{rhs}(\mathsf{Y}))\big[\mathit{pars}(\mathsf{Y}) := \overrightarrow{t}\big]\Big)$$

$$\mathit{guard}(S, \mathsf{Y}(\overrightarrow{t}) \mathbin{{}^{\varsigma}} t) = \mathit{simpl}(\mathit{guard}(S, \mathsf{Y}(\overrightarrow{t})) \mathbin{{}^{\varsigma}} t)$$

$$\mathit{guard}(S, p_1 \cdot p_2) = \mathit{rewr}\Big(\mathit{simpl}\big(\mathit{guard}(S, p_1) \cdot p_2\big)\Big)$$

$$\mathit{guard}(S, \rho_R \circ \tau_I \circ \partial_H(p)) = \mathit{rewr}\big(\rho_R \circ \tau_I \circ \partial_H(\mathit{guard}(S,p))\big)$$

$$\begin{aligned}
guard\big(S, p_1 \parallel p_2\big) = rewr\Big(& simpl'\big(guard(S,p_1) \mathbin{\|\!_} p_2, simpl1\,(\partial\mathcal{U}(guard(S,p_2)))\big)\\
&+ simpl'\big(guard(S,p_2) \mathbin{\|\!_} p_1, simpl1\,(\partial\mathcal{U}(guard(S,p_1)))\big)\\
&+ simpl\big(guard(S,p_1) \mid guard(S,p_2)\big)\Big)\\
guard\big(S, (p_1 \parallel p_2) \,{}^{\varsigma}\, t\big) = simpl(& guard\big(S, p_1 \parallel p_2\big) \,{}^{\varsigma}\, t)
\end{aligned}$$

Here we use the function *rewr* from Subsection 6.2.2 and the function S_0 from Section 4.2.1, which renames variables that are bound more than once. The functions *simpl* and *simpl′* are defined in Appendix B.3.1. It shows that for any term q^1 and q^2 in the form of the right-hand side of an equation in TPEGNF, and for any term p having syntax (6.3) we can transform $q^1 \cdot p$, $q^1 \mathbin{\|\!\_} p$, $q^1 \mid q^2$, $q^1 \,{}^{\varsigma}\, t$ and $t \ggg q^1$ to the form of the right-hand side of an equation in TPEGNF by applying the axioms of timed μCRL.

The function *simpl1* simplifies terms of the form $\partial\mathcal{U}(p)$, where p has the form of the right-hand side of a TPEGNF equation. It is needed for the elimination of $\ll$ from the expansions of the left merge. The result has a simple form which allows to eliminate $\ll$ operations from terms like $\mathsf{a}(\overrightarrow{d}) \ll p$.

$$\begin{aligned}
simpl1\Bigg(\partial\mathcal{U}\Big(&\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \cdot p_i \lhd c_i \rhd \delta{}^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \lhd c_j \rhd \delta{}^{\varsigma}\mathbf{0}\\
&+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0}\Big)\Bigg)\\
&= \sum_{k\in I\cup J\cup\{\delta\}}\ \sum_{\overrightarrow{e_k:E_k}} \delta \,{}^{\varsigma}\, t_k \lhd c_k \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned}$$

Proposition 6.2.7. *For any term p having the form of the right-hand side of a TPEGNF equation the transformation performed by function simpl1 is derivable from the axioms of timed μCRL.*

Proof. See Appendix B.3.3. □

Proposition 6.2.8. *For any terms q_1 and q_2 of the form of the right-hand side of a TPEGNF equation the transformation performed by function simpl is derivable from the axioms of timed μCRL.*

Proof. See Appendix B.3.2. □

Proposition 6.2.9. *For any finite system G_3 in pre-TPEGNF with acyclic PNUDG, and any process name X in it, the function guard is well-defined on rhs(X, G_3).*

Proof. Let n be the number of equations in G_3. The only clause that makes the argument of *guard* larger is the third one. Due to the fact that PNUDG is acyclic, this rule cannot be applied more than n times deep (otherwise for some process name Z we would have a cycle). □

We define the system G_4 in the following way. For each equation

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta^{\varsigma}\mathbf{0} \quad \text{in } G_3 \text{ we add}$$

$$\mathsf{X}(\overrightarrow{d{:}D}) = \mathit{guard}\Big(\{\overrightarrow{d}\}, \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta^{\varsigma}\mathbf{0}\Big) \quad \text{to } G_4.$$

Lemma 6.2.10. *The equations in G_4 are in TPEGNF.*

Proof. Due to Proposition 6.2.9 we can apply induction on the definition of *guard*. The second through fifth clause of the *guard* definition are the induction base and they are trivially in TPEGNF. The sixth clause is also trivial. In the first clause the only rules in Tables 6.3, 6.4 and 6.5 that can be applied are (RCOND7T), (RSUM12T), (RCOND4T) and (RSUM4), which bring the right-hand side to the desired form.

For the nineth clause *rewr* can be applied with all the rules for renaming, hiding and encapsulation, which preserve TPEGNF. For the remaining clauses we use the fact that both *simpl* and *simpl′* produce terms in TPEGNF. □

Proposition 6.2.11. *Let G_3 and G_4 be defined as above. Then $G_3 = G_4$.*

Proof. It was already noted before that the transformations performed by *rewr* and S_0 are derivable from the axioms of timed μCRL. It is shown in Appendix B.3 that the transformations performed by *simpl*, *simpl′* and *simpl1* are derivable from the axioms as well. Due to Lemma 3.5.3 and Lemma 3.5.1, all transformations performed by *guard* lead to equivalent systems. We note that care has been taken to rename some data variables during the substitution (in the third clause of *guard* definition) in order to make the substitution and the following applications of the axioms sound. □

As in the untimed case, the transformation performed in this step does not increase the number of equations, but their sizes may grow exponentially, due to applications of (A4). An example of such exponential growth can be found in Section 4.2.4. We also note that similar growth is possible due to application of axioms (CM4) for the left merge, and (CM8) and (CM9) for communication.

6.2.5 Elimination of Time-Related Operations

In this subsection we show how to transform our system of equations into post-TPEGNF. This is done by first making all of the equations well-timed and then applying a modified guarding procedure that preserves well-timedness. Well-timedness of the equations is important not only for elimination of time related operations, but also for further modeling of parallel and sequential composition by a data type (cf. Section 6.3).

Well-Timed Equations

First we define a notion of well-timedness and show that many operations preserve well-timedness without introduction of new $\gg$ operations. Then we define a modified guarding procedure that preserves well-timedness.

Definition 6.2.12. A term $\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \cdot p$ is *well-timed* if $p \approx t \gg p$. If t is such that $c(t) \approx \mathbf{t}$ implies $p \approx t \gg p$, then $\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \cdot p \lhd c(t) \rhd \delta^{\varsigma}\mathbf{0}$ is also well-timed. Terms $\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t$ and $\delta \,^{\varsigma}\, t$ are also well-timed. If p and q are well-timed terms, then $p+q$, $\sum_{d:D} p$ and $p \lhd c \rhd \delta^{\varsigma}\mathbf{0}$ are also well-timed terms.

An equation in TPEGNF is well-timed if for all $i \in I$ the terms $\mathsf{a}_i(\overrightarrow{t_i})^{\varsigma}t_i{\cdot}p_i{\lhd}c_i{\rhd}\delta^{\varsigma}\mathbf{0}$ are well-timed. A system of equations is well-timed if all of the equations in it are.

Any equation in TPEGNF can be made well-timed by replacing the terms p_i with $t_i \gg p_i$. The operation $\gg$ can be "pushed" inside sequential composition and the renaming operations. However, it cannot be pushed inside $\|$ and $^{\varsigma}$ operations. For example, $5 \gg (\mathsf{a} \,^{\varsigma}\, 5 \,\|\, \delta \,^{\varsigma}\, 2) \approx \delta \,^{\varsigma}\, 5$, and $(5 \gg \mathsf{a} \,^{\varsigma}\, 5) \,\|\, (5 \gg \delta \,^{\varsigma}\, 2) \approx \mathsf{a} \,^{\varsigma}\, 5 \cdot \delta \,^{\varsigma}\, 5$. Another example would be $5 \gg (\mathsf{a} \,^{\varsigma}\, 4) \approx \delta \,^{\varsigma}\, 5$ and $(5 \gg \mathsf{a}) \,^{\varsigma}\, 4 \approx \delta \,^{\varsigma}\, 4$.

From the definition of *simpl* it is clear that expanding $q \,^{\varsigma}\, t$, $t \gg q$ and $t \ggg q$, where q has the form of the right-hand side of a well-timed equation in TPEGNF, preserves well-timedness and does not introduce new $^{\varsigma}$, $\gg$ and $\ggg$ operations to the terms p_i of q.

It is harder to define the parallel composition in a way that preserves well-timedness without introduction of new $\gg$ operations. This can be done by introducing $\ggg$ operations instead. The following properties can be obtained from Lemma B.2.4.38&.39:

$$\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \,⫗\, q \approx (\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \ll q) \cdot t \ggg q$$

and if $p \approx t \gg p$, then

$$(\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \cdot p) \,⫗\, q \approx (\mathsf{a}(\overrightarrow{t}) \,^{\varsigma}\, t \ll q) \cdot (p \,\|\, t \ggg q)$$

In Lemma B.3.4 it is shown that for every completely guarded term q the right-hand sides of the above identities rewrite to well-timed terms.

Using these two identities and the definition of *simpl* for $⫗$ and $|$ it is easy to see that parallel composition of two well-timed equations can be turned into a well-timed equation as well. For $⫗$ we need to change *simpl*$'$ using the identities above. For $|$ the function *simpl* preserves well-timedness because:

- both communicating parts are well-timed (assumption),
- communicating actions occur at the same time (axioms (ATA7), (ATA8) and (ATA1$'$)),
- if $t \gg x \approx x$ and $t \gg y \approx y$, then $t \gg (x \,\|\, y) \approx x \,\|\, y$ (Lemma B.2.4.33).

It is also clear that renaming operations preserve well-timedness. The only remaining operation is sequential composition. As can be seen from the definition of *simpl* for sequential composition (in Appendix B.3.1), the terms $\mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \cdot p$ are not necessarily well-timed, so they should be replaced by $\mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \cdot t_j \gg p$ to maintain well-timedness. So, what happens here is that we actually introduce $\gg$, but, as will be shown later, in our procedure the term p decreases at each step.

The function $\mathit{guard1} : \mathit{DVar} \times \mathit{Terms}(|G|) \rightarrow \mathit{Terms}(|G|)$ is similar to *guard* but makes sure that the result is well-timed. It has a definition that is similar to the one of *guard*, with differences only in the following cases:

$$\begin{aligned}
&\mathit{guard1}(S, p_1 \cdot p_2) = \mathit{rewr}\Big(\mathit{simpl2}\big(\mathit{guard1}(S,p_1) \cdot p_2\big)\Big)\\
&\mathit{guard1}\big(S, p_1 \parallel p_2\big) = \mathit{rewr}\Big(\mathit{simpl2'}\big(\mathit{guard1}(S,p_1) \mathbin{⫽} p_2, \mathit{simpl1}(\partial\mathcal{U}(\mathit{guard1}(S,p_2)))\big)\\
&\qquad\qquad + \mathit{simpl2'}\big(\mathit{guard1}(S,p_2) \mathbin{⫽} p_1, \mathit{simpl1}(\partial\mathcal{U}(\mathit{guard1}(S,p_1)))\big)\\
&\qquad\qquad + \mathit{simpl}\big(\mathit{guard1}(S,p_1) \mid \mathit{guard1}(S,p_2)\big)\Big)\\
&\mathit{guard1}(S, t \gg p) = \mathit{rewr}(\mathit{guard1}(S, \mathit{rewr1}(t \ggg p)) + \delta \,{}^{\varsigma}\, t)\\
&\mathit{guard1}(S, t \ggg p) = \mathit{simpl}(t \ggg \mathit{guard1}(S,p))
\end{aligned}$$

The partial functions *simpl2* and *simpl2′* are defined as follows.

$$\begin{aligned}
&\mathit{simpl2}\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \cdot p_i \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}\Big) \cdot p\Bigg)\\
&\quad = \sum_{i\in I}\sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \cdot \big(p_i \cdot p\big) \lhd c_i \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad + \sum_{j\in J}\sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \cdot \mathit{rewr1}(t_j \gg p) \lhd c_j \rhd \delta^{\varsigma}\mathbf{0} + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\mathit{simpl2'}\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \cdot p_i \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}\Big) \mathbin{⫽} p, p'\Bigg)\\
&\quad = \sum_{i\in I}\sum_{\overrightarrow{e_i : E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \ll p'\big) \cdot \big(p_i \parallel \mathit{rewr1}(t_i \ggg p)\big) \lhd c_i \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad + \sum_{j\in J}\sum_{\overrightarrow{e_j : E_j}} \big(\mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \ll p'\big) \cdot \mathit{rewr1}(t_j \ggg p) \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad + \sum_{\overrightarrow{e_\delta : E_\delta}} \big(\delta \,{}^{\varsigma}\, t_\delta \ll p'\big) \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}
\end{aligned}$$

Correctness of these definitions follows from the correctness of the corresponding definitions of *simpl* and *simpl'*, and from the axiom (AT2) and Lemma B.3.4.

The function *rewr1* is defined as a result of applying the rewrite system consisting of the rules in Table 6.7. Termination of this rewrite system is trivial (using the RPO "$\ggg$" > "$\gg$" > other operations).

Lemma 6.2.13. *The rules in Table 6.7 are derivable from the axioms of timed μCRL.*

Proof. See Lemma B.2.4 in Appendix B.2.2. $\square$

$t \gg \delta \to \delta$	(RATB1')
$t \gg \delta \cdot\!\!\!\!^{\varsigma}\, u \to \delta \,^{\varsigma}\, max(t,u)$	(RATB1'')
$t \gg (x \cdot y) \to (t \gg x) \cdot y$	(RATB3)
$t \gg (\partial_H(x)) \to \partial_H(t \gg x)$	(RD8)
$t \gg (\tau_I(x)) \to \tau_I(t \gg x)$	(RT8)
$t \gg (\rho_R(x)) \to \rho_R(t \gg x)$	(RR8)
$t \gg (u \gg x) \to max(t,u) \gg x$	(RATB7)
$t \ggg \delta \to \delta$	(RATD1')
$t \ggg (x \cdot y) \to (t \ggg x) \cdot y$	(RATD3)
$t \ggg (x \parallel y) \to (t \ggg x) \parallel (t \ggg y)$	(RATD8)
$t \ggg (\partial_H(x)) \to \partial_H(t \ggg x)$	(RD9)
$t \ggg (\tau_I(x)) \to \tau_I(t \ggg x)$	(RT9)
$t \ggg (\rho_R(x)) \to \rho_R(t \ggg x)$	(RR9)
$t \ggg (x \,^{\varsigma}\, u) \to (t \ggg x) \,^{\varsigma}\, u$	(RATD6)
$t \ggg (u \gg x) \to u \gg (t \ggg x)$	(RATD9)
$t \ggg (u \ggg x) \to max(t,u) \ggg x$	(RATD7)

Table 6.7: Rewrite rules defining *rewr1*.

According to the intuition presented above, we state without further proof that the modified simplification functions preserve well-timedness.

Proposition 6.2.14. *Let well-timed terms q^1, q^2 be in the form of the right-hand side of an equation in TPEGNF, and let term p be in the form of syntax* (6.3). *Let $p' = \partial\mathcal{U}(p)$ and let time term t be such that the bound variables of q^1, q^2 and p' do not occur freely in p and t. Then the results of the expressions listed below rewrite to well-timed terms.*

1. *$simpl2(q^1 \cdot p)$;*
2. *$simpl2'(q^1 \mathbin{|\!\lfloor} p, p')$;*
3. *$simpl(q^1 \mid q^2)$;*
4. *$simpl(q^1 \,^{\varsigma}\, t)$;*
5. *$simpl(t \ggg q^1)$.*

Transformation to Refined-TPEGNF

As input we take a system G_4 of equations in TPEGNF, and as a result we produce an equivalent system of well-timed equations in refined-TPEGNF.

1. First we make the system well-timed by replacing p_i in G_4 with $rewr1(t_i \gg p_i)$. We get a system G'.

2. Each subterm of the form $t \gg (p \parallel q)$ and $(p \parallel q) \ll t$ in p_i from G' is replaced by a new process name Y, and we add the equation for Y. As a result, we get a system G'' consisting of two parts:

   - the set G''_1 contains the equations from G' with all the $t \gg (p \parallel q)$ and $(p \parallel q) \ll t$ replaced;

   - the set G''_2 contains the equations of the form $\mathsf{Y}(\overrightarrow{d}) = p_\mathsf{Y}$, where p_Y has the syntax of (6.3). Moreover, G''_2 contains no internal dependency loops.

3. We use *guard1* to make the equations in G''_2 guarded. By doing so we generate $\ggg$ operations, and $\gg$ operations when unfolding sequential composition. We generate no $\ll$ operations.

4. We iterate steps 2-3 till we get rid of all $\gg$ operations. The number of newly created $\gg$ operations is limited by the maximal number of sequential compositions in terms p_i from G. From the fact that we eliminated all $t \gg (p \parallel q)$ and $(p \parallel q) \ll t$ constructs, and from the properties of *rewr1* it is clear that we obtain a system of equations G_5 in refined-TPEGNF.

Proposition 6.2.15. *For any system $G''_1 \cup G''_2$ obtained on any iteration of the above mentioned procedure the function guard1 is well-defined on the right-hand side of any equation.*

Proposition 6.2.16. *Let G_4 and G_5 be defined as above. Then $G_4 \overset{cond}{\Rightarrow} G_5$.*

The following example shows how the iterations are performed.

Example 6.2.17. We start with one equation for X with no parameters:

$$\mathsf{X} = \mathsf{a} \ll 1 \cdot (\mathsf{X} \parallel \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) + \mathsf{b} \ll 2$$

First we make the equation well-timed by changing $(\mathsf{X}\parallel\mathsf{X}\cdot(\mathsf{X}\parallel\mathsf{X}))$ to $1\gg(\mathsf{X}\parallel\mathsf{X}\cdot(\mathsf{X}\parallel\mathsf{X}))$. After that we introduce a new equation for the former term. We get the following system:

$$\mathsf{X} = \mathsf{a} \ll 1 \cdot 1 \gg \mathsf{Y} + \mathsf{b} \ll 2$$
$$\mathsf{Y} = \mathsf{X} \parallel \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})$$

Next, we apply *guard1* to the right-hand side of Y:

$$
\begin{aligned}
&\mathsf{Y} = \mathsf{X} \mathbin{\|\!\lfloor} \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X}) + (\mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) \mathbin{\|\!\lfloor} \mathsf{X} \\
&\mathsf{Y} = (\mathsf{a} \cdot\!\! 1 \cdot 1 \gg \mathsf{Y} + \mathsf{b} \cdot\!\! 2) \mathbin{\|\!\lfloor} \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X}) + ((\mathsf{a} \cdot\!\! 1 \cdot 1 \gg \mathsf{Y} + \mathsf{b} \cdot\!\! 2) \cdot (\mathsf{X} \parallel \mathsf{X})) \mathbin{\|\!\lfloor} \mathsf{X} \\
&\mathsf{Y} = \mathsf{a} \cdot\!\! 1 \cdot (1 \gg \mathsf{Y} \parallel 1 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) + \mathsf{b} \cdot\!\! 2 \cdot 2 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X}) \\
&\qquad\qquad + (\mathsf{a} \cdot\!\! 1 \cdot 1 \gg \mathsf{Y} \cdot (\mathsf{X} \parallel \mathsf{X}) + \mathsf{b} \cdot\!\! 2 \cdot 2 \gg (\mathsf{X} \parallel \mathsf{X})) \mathbin{\|\!\lfloor} \mathsf{X} \\
&\mathsf{Y} = \mathsf{a} \cdot\!\! 1 \cdot (1 \gg \mathsf{Y} \parallel 1 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) + \mathsf{b} \cdot\!\! 2 \cdot 2 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X}) \\
&\qquad\qquad + \mathsf{a} \cdot\!\! 1 \cdot ((1 \gg \mathsf{Y} \cdot (\mathsf{X} \parallel \mathsf{X})) \parallel 1 \ggg \mathsf{X}) + \mathsf{b} \cdot\!\! 2 \cdot (2 \gg (\mathsf{X} \parallel \mathsf{X}) \parallel 2 \ggg \mathsf{X})
\end{aligned}
$$

Now we see that we need to introduce a new equation for $\mathsf{X} \parallel \mathsf{X}$ because we need to eliminate $2 \gg (\mathsf{X} \parallel \mathsf{X})$ from the right-hand side. Thus we proceed to the next iteration by adding the new process equation for Z and applying guarding to it.

$$
\begin{aligned}
&\mathsf{Y} = \mathsf{a} \cdot\!\! 1 \cdot (1 \gg \mathsf{Y} \parallel 1 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) + \mathsf{b} \cdot\!\! 2 \cdot 2 \ggg \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X}) \\
&\qquad\qquad + \mathsf{a} \cdot\!\! 1 \cdot ((1 \gg \mathsf{Y} \cdot (\mathsf{X} \parallel \mathsf{X})) \parallel 1 \ggg \mathsf{X}) + \mathsf{b} \cdot\!\! 2 \cdot (2 \gg \mathsf{Z} \parallel 2 \ggg \mathsf{X}) \\
&\mathsf{Z} = \mathsf{X} \parallel \mathsf{X} \\
&\mathsf{Z} = \mathsf{X} \mathbin{\|\!\lfloor} \mathsf{X} \\
&\mathsf{Z} = (\mathsf{a} \cdot\!\! 1 \cdot 1 \gg \mathsf{Y} + \mathsf{b} \cdot\!\! 2) \mathbin{\|\!\lfloor} \mathsf{X} \\
&\mathsf{Z} = \mathsf{a} \cdot\!\! 1 \cdot (1 \gg \mathsf{Y} \parallel (1 \ggg \mathsf{X})) + \mathsf{b} \cdot\!\! 2 \cdot 2 \ggg \mathsf{X}
\end{aligned}
$$

It is easy to see that the iteration in this example resulted from the nesting of parallel and sequential compositions in $\mathsf{X} \parallel \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})$. In general, if we replace this term by the term $\mathsf{X} \parallel (\mathsf{X} \cdot (\mathsf{X} \parallel (\mathsf{X} \cdot \ldots (\mathsf{X} \parallel \mathsf{X}) \cdots)))$ with n nestings of parallel and sequential compositions, then we will need n iterations to make this term well-timed.

It is interesting that n sequential compositions will lead to no iterations as $t \gg (\mathsf{X} \cdot \mathsf{X} \cdot \ldots \cdot \mathsf{X}) \approx (t \gg \mathsf{X}) \cdot \mathsf{X} \cdot \ldots \cdot \mathsf{X}$. Similarly, n parallel compositions will lead to only one iteration because $\mathsf{X} \parallel \mathsf{X} \parallel \ldots \parallel \mathsf{X}$ will be replaced by a new name Y. Applying *guard1* to Y will lead to $n-1$ applications of *guard1* to parallel compositions, which do not introduce new $\gg$ operations.

Transformation to Post-TPEGNF

As the next step we take every term of G_5 that has the syntax of p_1 from (6.4) and apply *rewr2* to it, which is *rewr1* extended with the following rules:

$$
\begin{aligned}
t \gg x &\to t \ggg x + \delta \cdot\!\! t && \text{(RATB0)} \\
(x + y) \cdot\!\! t &\to x \cdot\!\! t + y \cdot\!\! t && \text{(RATA2)} \\
\delta \cdot\!\! t \cdot\!\! u &\to \delta \cdot\!\! \mathit{min}(t, u) && \text{(RATAD1)} \\
\delta \cdot\!\! t + \delta \cdot\!\! u &\to \delta \cdot\!\! \mathit{max}(t, u) && \text{(RA6}'\text{)}
\end{aligned}
$$

As a result we get terms of the following syntax, where a term p can be written as $\mathbf{0} \ggg p$:

$$
p_1' ::= \mathsf{a}(\vec{t}) \mid \delta \mid \mathsf{Y}(\vec{t}) \mid \delta \cdot\!\! t \mid (t \ggg \mathsf{a}(\vec{t})) \cdot\!\! u_1 \cdots \cdot\!\! u_n + \delta \cdot\!\! w \mid (t \ggg \mathsf{Y}(\vec{t})) \cdot\!\! u_1 \cdots \cdot\!\! u_n + \delta \cdot\!\! w
$$

We introduce a new process name $\mathsf{X_a}$ for each action a that occurs inside the process terms p_i not in scope of $\cdot$, with parameters corresponding to those of the action (and a new process name X_δ for δ). Thus we add equations $\mathsf{X_a}(\overrightarrow{d_\mathsf{a}{:}D_\mathsf{a}}) = \sum_{u:\mathit{Time}} \mathsf{a}(\overrightarrow{d_\mathsf{a}}) \cdot u$ and $\mathsf{X}_\delta = \sum_{u:\mathit{Time}} \delta \cdot u$ for a fresh variable u to the system, and replace the occurrences of actions $\mathsf{a}(\overrightarrow{t})$ by $\mathsf{X_a}(\overrightarrow{t})$, and δ by X_δ. In the same way we add $\mathsf{X}_\delta(t{:}\mathit{Time}) = \delta \cdot t$ to replace $\delta \cdot t$, and

$$\begin{aligned}\mathsf{X}_{\mathsf{a}_n}(\overrightarrow{d_\mathsf{a}{:}D_\mathsf{a}}, t{:}\mathit{Time}, u_1{:}\mathit{Time}, \ldots, u_n{:}\mathit{Time}) = \\ \mathsf{a}(\overrightarrow{d_\mathsf{a}}) \cdot u_1 \lhd t \le u_1 \wedge \mathit{eq}(u_1, u_2) \wedge \cdots \wedge \mathit{eq}(u_{n-1}, u_n) \rhd \delta \cdot \mathbf{0} \\ + \delta \cdot \mathit{max}(w, \mathit{min}(u_1, \ldots, u_n))\end{aligned}$$

to replace $(t \ggg \mathsf{a}(\overrightarrow{t})) \cdot u_1 \cdots \cdot u_n + \delta \cdot w$. The last syntactic category is replaced by applying *guard1* to get rid of the unguarded occurrence of $\mathsf{Y}(\overrightarrow{t})$.

Proposition 6.2.18. *Let the system G_6 of process equations be obtained after processing the system G_5 as described above. Then for all $\mathsf{X} \in |G_5|$ we have $(\mathsf{X}, G_6) = (\mathsf{X}, G_5)$ and G_6 is in post-TPEGNF.*

To illustrate this transformation we continue with the system from Example 6.2.17.

Example 6.2.19. The resulting system of Example 6.2.17 (taking into account that $1 \ggg \mathsf{X} \approx \mathsf{X}$ and $1 \gg \mathsf{Y} \approx \mathsf{Y}$) will look as follows:

$$\begin{aligned}\mathsf{X} &= \mathsf{a} \cdot 1 \cdot \mathsf{Y} + \mathsf{b} \cdot 2 \\ \mathsf{Y} &= \mathsf{a} \cdot 1 \cdot (\mathsf{Y} \parallel \mathsf{X} \cdot (\mathsf{X} \parallel \mathsf{X})) + \mathsf{b} \cdot 2 \cdot \mathsf{X}_2 \cdot (\mathsf{X} \parallel \mathsf{X}) \\ &\quad + \mathsf{a} \cdot 1 \cdot ((\mathsf{Y} \cdot (\mathsf{X} \parallel \mathsf{X})) \parallel \mathsf{X}) + \mathsf{b} \cdot 2 \cdot (\mathsf{Z}_2 \parallel \mathsf{X}_2) \\ \mathsf{Z} &= \mathsf{a} \cdot 1 \cdot (\mathsf{Y} \parallel \mathsf{X}) + \mathsf{b} \cdot 2 \cdot \mathsf{X}_2 \\ \mathsf{X}_2 &= \mathsf{b} \cdot 2 \\ \mathsf{Z}_2 &= \mathsf{b} \cdot 2 \cdot \mathsf{X}_2\end{aligned}$$

Here we introduced X_2 and Z_2 for $2 \ggg \mathsf{X}$ and $2 \ggg \mathsf{Z}$, respectively.

6.3 Reusing LM and ML Data Types for Timed μCRL

At this point we can perform parameter harmonization and make one process equation out of our system as it was done in Section 4.3. It is clear that these transformations preserve well-timedness. Thus we start from the system G_7 consisting of a single well-timed process equation for X in post-TPEGNF (see Definition 6.2.1).

6.3.1 Maximal Delay in a State

For the translation we need to know if the process X in a given state $\overrightarrow{t}$ can delay till given time t. By applying the $\partial\mathcal{U}$ operation to the equation for X we get the following

(Proposition B.3.1):

$$\partial\mathcal{U}(\mathsf{X}(\overrightarrow{t})) \approx \sum_{k\in I\cup J\cup\{\delta\}} \sum_{\overrightarrow{e_k:E_k}} \delta \,{}^{\varsigma}\, t_k(\overrightarrow{t,e_k}) \lhd c_k(\overrightarrow{t,e_k}) \rhd \delta{}^{\varsigma}\mathbf{0}$$

According to Proposition B.3.2 if we assume that $I \cup J \cup \{\delta\} = \{0, \dots, n\}$, then the identity above is equivalent to

$$\partial\mathcal{U}(\mathsf{X}(\overrightarrow{t})) \approx \sum_{u:Time} \sum_{\overrightarrow{e_0:E_0}} \cdots \sum_{\overrightarrow{e_n:E_n}} \delta \,{}^{\varsigma}\, u \lhd \bigvee\nolimits_{0\le i\le n} (u \le t_i(\overrightarrow{t,e_i}) \wedge c_i(\overrightarrow{t,e_i})) \rhd \delta{}^{\varsigma}\mathbf{0}$$

Now, if we define new data type $\overrightarrow{E} = \overrightarrow{Time, E_0, \dots, E_n}$ and the functions $u : \overrightarrow{E} \to Time$ and $c : \overrightarrow{D_{\mathsf{X}}, E} \to Bool$ as

$$u(w, \overrightarrow{e_0}, \dots, \overrightarrow{e_n}) \approx w$$

$$c(\overrightarrow{t}, w, \overrightarrow{e_0}, \dots, \overrightarrow{e_n}) \approx \bigvee\nolimits_{0\le i\le n} (w \le t_i(\overrightarrow{t,e_i}) \;\wedge\; c_i(\overrightarrow{t,e_i}))$$

our identity will look like

$$\partial\mathcal{U}(\mathsf{X}(\overrightarrow{t})) \approx \sum_{\overrightarrow{e:E}} \delta \,{}^{\varsigma}\, u(\overrightarrow{e}) \lhd c(\overrightarrow{t,e}) \rhd \delta{}^{\varsigma}\mathbf{0}$$

which basically means that $\mathsf{X}(\overrightarrow{t})$ can delay till time t if for some $\overrightarrow{e{:}E}$ such that $c(\overrightarrow{t,e}) \approx \mathbf{t}$ we have $t \le u(\overrightarrow{e})$.

Additionally, for dealing with parallel compositions of the form $\mathsf{X}(\overrightarrow{t_0}) \parallel \cdots \parallel \mathsf{X}(\overrightarrow{t_m})$, we need to know if the process X can delay till a given time t in *all* given states $\overrightarrow{t_0}, \dots, \overrightarrow{t_m}$. The process that can delay as long as *all* of the processes $\mathsf{X}(\overrightarrow{t_0}), \dots, \mathsf{X}(\overrightarrow{t_m})$ can delay is expressed as the following term:

$$\begin{aligned}
&\partial\mathcal{U}(\mathsf{X}(\overrightarrow{t_0}) \parallel \cdots \parallel \mathsf{X}(\overrightarrow{t_m}))\\
&\qquad \approx \partial\mathcal{U}(\mathsf{X}(\overrightarrow{t_0})) \parallel \cdots \parallel \partial\mathcal{U}(\mathsf{X}(\overrightarrow{t_m}))\\
&\qquad \approx \sum_{t:Time} \sum_{\overrightarrow{e_0:E}} \cdots \sum_{\overrightarrow{e_m:E}} \delta \,{}^{\varsigma}\, t \lhd \bigwedge\nolimits_{0\le i\le m} (t \le u(\overrightarrow{e_i}) \wedge\; c(\overrightarrow{t_i, e_i})) \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned}$$

If m is not known apriori, as in our case, we can reuse the data type *LState* representing lists of state vectors (cf. Section 5.2.3), and define the data type *LE* to represent lists of elements of sort $\overrightarrow{E}$. The process that can delay as long as *all* of the processes $\mathsf{X}(\overrightarrow{t_i})$ can, where $\overrightarrow{t_i}$ are elements of a list of state vectors $\overrightarrow{ls}{:}LState$, can be expressed as the following term:

$$\sum_{t:Time} \sum_{\overrightarrow{le:LE}} \delta \,{}^{\varsigma}\, t \lhd len(\overrightarrow{le}) = len(\overrightarrow{ls}) \wedge \bigwedge\nolimits_{0\le i\le len(\overrightarrow{ls})} (t \le u(\overrightarrow{e_i}) \wedge\; c(\overrightarrow{t_i, e_i})) \rhd \delta{}^{\varsigma}\mathbf{0} \quad (6.6)$$

To bring this within the syntax of timed μCRL ($len(\overrightarrow{ls})$ is not a constant, so the conjunction for all i s.t. $0 \le i \le len(\overrightarrow{ls})$ is not an abbreviation) we need the function $sat\_mmd : Time \times LState \times LE \to Bool$ defined as

$$sat\_mmd(t, \langle\overrightarrow{t_0}, \dots, \overrightarrow{t_m}\rangle, \langle\overrightarrow{e_0}, \dots, \overrightarrow{e_m}\rangle) = \bigwedge\nolimits_{0\le i\le m} (t \le u(\overrightarrow{e_i}) \wedge\; c(\overrightarrow{t_i, e_i}))$$

In μCRL this function could be specified as follows:

$$sat\_mmd(t, \langle\rangle, \langle\rangle) \approx \mathbf{t}$$

$$sat\_mmd(t, add(\overrightarrow{t}, \overrightarrow{ls}), add(\overrightarrow{e}, \overrightarrow{le})) \approx t \leq u(\overrightarrow{e}) \wedge\ c(\overrightarrow{t,e}) \wedge\ sat\_mmd(t, ls, \overrightarrow{le})$$

For a way of implementing this function and other mentioned data types in timed μCRL we refer to Appendix C.7.

With the introduction of *sat_mmd* the term (6.6) will look as follows:

$$\sum_{t:Time} \sum_{\overrightarrow{le:LE}} \delta \,{}^{\varsigma}\, t \lhd len(\overrightarrow{le}) = len(\overrightarrow{ls}) \wedge sat\_mmd(t, \overrightarrow{ls}, \overrightarrow{le}) \rhd \delta{}^{\varsigma}\mathbf{0}.$$

6.3.2 Final TLPE Definition

The case of timed μCRL differs from the untimed case in the following aspects.

- The resulting TLPE Z has an extra parameter $t{}^{\varsigma}\, Time$ to keep track of "current" time.
  - An extra condition must be satisfied: no action must be executed earlier than the "current" time.
  - After an action is executed at time u, u becomes the "current" time for the TLPE Z.
- An extra condition must be satisfied: if n components are ready to be executed and $0 < k < n$ of them communicate by executing an action at time u, then all other ready components must be able to delay till u.
- TLPE Z can delay as long as all ready components could, but not less than till the "current" time.

For the case with handshaking and no renaming operations we start with the following well-timed equation for X.

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d{:}D}) = & \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) \,{}^{\varsigma}\, t_i(\overrightarrow{d,e_i}) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0} \\
& + \sum_{j\in J} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \,{}^{\varsigma}\, t_j(\overrightarrow{d,e_j}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta{}^{\varsigma}\mathbf{0} \\
& + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t_\delta(\overrightarrow{d,e_\delta}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned}$$

where I and J are disjoint, and all $p_i(\overrightarrow{d,e_i})$ have the following syntax:

$$p ::= \mathsf{X}(\overrightarrow{t}) \mid p \cdot p \mid p \parallel p \qquad (6.7)$$

We can now define a system L consisting of process equation Z, which mimics behavior of X, in the following way:

$$
\begin{aligned}
&\mathsf{Z}(t{:}\mathit{Time}, \mathit{lm}{:}\mathit{LM}) = \\
&\quad \sum_{i\in I} \sum_{n:\mathit{Nat}} \sum_{\overrightarrow{e_i:E_i}} \sum_{\overrightarrow{le:LE}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i})) \,{}^{\varsigma}\, t_i(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}) \\
&\qquad\quad \cdot \mathsf{Z}(t_i(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}), \mathit{replf1}(\mathit{lm},n,\mathbf{mklm}_i[p_i](\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}))) \\
&\qquad\qquad \lhd\, n < \mathit{lenf}(\mathit{lm}) \wedge c_i(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}) \\
&\qquad\qquad\quad \wedge t \le t_i(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}) \wedge \mathit{len}(\overrightarrow{\mathit{le}}) + 1 = \mathit{lenf}(\mathit{lm}) \\
&\qquad\qquad\quad \wedge \mathit{sat\_mmd}(t_i(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_i}), \overrightarrow{\mathit{getfn}(\mathit{lm},\mathit{rem}(n,\mathit{ln}))}, \overrightarrow{\mathit{le}}) \rhd \delta\,{}^{\varsigma}\mathbf{0} \\
&\quad + \sum_{j\in J} \sum_{n:\mathit{Nat}} \sum_{\overrightarrow{e_j:E_j}} \sum_{\overrightarrow{le:LE}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j})) \,{}^{\varsigma}\, t_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \cdot \mathsf{Z}(\mathit{remf1}(\mathit{lm},n)) \\
&\qquad\qquad \lhd\, n < \mathit{lenf}(\mathit{lm}) \wedge \mathit{remf1}(\mathit{lm},n) \ne \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \\
&\qquad\qquad\quad \wedge t \le t_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \wedge \mathit{len}(\overrightarrow{\mathit{le}}) + 1 = \mathit{lenf}(\mathit{lm}) \\
&\qquad\qquad\quad \wedge \mathit{sat\_mmd}(t_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}), \overrightarrow{\mathit{getfn}(\mathit{lm},\mathit{rem}(n,\mathit{ln}))}, \overrightarrow{\mathit{le}}) \rhd \delta\,{}^{\varsigma}\mathbf{0} \\
&\quad + \sum_{j\in J} \sum_{n:\mathit{Nat}} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j})) \,{}^{\varsigma}\, t_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \\
&\qquad\qquad \lhd\, n < \mathit{lenf}(\mathit{lm}) \wedge \mathit{remf1}(\mathit{lm},n) = \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \\
&\qquad\qquad\quad \wedge t \le t_j(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_j}) \rhd \delta\,{}^{\varsigma}\mathbf{0} \\
&\quad + \sum_{u:\mathit{Time}} \sum_{\overrightarrow{le:LE}} \delta \,{}^{\varsigma}\, u \\
&\qquad\qquad \lhd\, \mathit{len}(\overrightarrow{\mathit{le}}) = \mathit{lenf}(\mathit{lm}) \wedge (u \le t \vee \mathit{sat\_mmd}(u, \overrightarrow{\mathit{getfn}(\mathit{lm},\mathit{ln})}, \overrightarrow{\mathit{le}})) \rhd \delta\,{}^{\varsigma}\mathbf{0} \\
&\quad + \sum_{(k,l)\in I\gamma I} \sum_{n:\mathit{Nat}} \sum_{m:\mathit{Nat}} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \sum_{\overrightarrow{le:LE}} \gamma(\mathsf{a}_k,\mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}))\,{}^{\varsigma}\, t_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}) \\
&\qquad\quad \cdot \mathsf{Z}(t_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}), \\
&\qquad\qquad\quad \mathit{replf2}(\mathit{lm},n,m,\mathbf{mklm}_k[p_k](\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}), \mathbf{mklm}_l[p_l](\overrightarrow{\mathit{getf1}(\mathit{lm},m),e'_l}))) \\
&\qquad\qquad \lhd\, n < m \wedge m < \mathit{lenf}(\mathit{lm}) \wedge \overrightarrow{f_k}(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1}(\mathit{lm},m),e'_l}) \\
&\qquad\qquad\quad \wedge c_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}) \wedge c_l(\overrightarrow{\mathit{getf1}(\mathit{lm},m),e'_l}) \\
&\qquad\qquad\quad \wedge t \le t_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}) \wedge t_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}) = t_l(\overrightarrow{\mathit{getf1}(\mathit{lm},m),e_l}) \\
&\qquad\qquad\quad \wedge \mathit{len}(\overrightarrow{\mathit{le}}) + 2 = \mathit{lenf}(\mathit{lm}) \\
&\qquad\qquad\quad \wedge \mathit{sat\_mmd}(t_k(\overrightarrow{\mathit{getf1}(\mathit{lm},n),e_k}), \overrightarrow{\mathit{getfn}(\mathit{lm},\mathit{rem}(n,\mathit{rem}(m,\mathit{ln})))}, \overrightarrow{\mathit{le}}) \rhd \delta\,{}^{\varsigma}\mathbf{0}
\end{aligned}
$$

$$
\begin{aligned}
&+\sum_{(k,l)\in I\gamma J}\sum_{n:Nat}\sum_{m:Nat}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E_l}}\sum_{\overrightarrow{le:LE}} \gamma(\mathsf{a}_k,\mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}))^{\text{c}}t_k(\overrightarrow{getf1(lm,n),e_k})\\
&\quad\cdot \mathsf{Z}(t_k(\overrightarrow{getf1(lm,n),e_k}), replremf2(lm,n,m,\mathbf{mklm}_k[p_k](\overrightarrow{getf1(lm,n),e_k})))\\
&\qquad \lhd\, n \neq m \wedge n < lenf(lm) \wedge m < lenf(lm)\\
&\qquad\ \wedge \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})\\
&\qquad\ \wedge c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l})\\
&\qquad\ \wedge t \leq t_k(\overrightarrow{getf1(lm,n),e_k}) \wedge t_k(\overrightarrow{getf1(lm,n),e_k}) = t_l(\overrightarrow{getf1(lm,m),e_l})\\
&\qquad\ \wedge len(\overrightarrow{le}) + 2 = lenf(lm)\\
&\qquad\ \wedge sat\_mmd(t_k(\overrightarrow{getf1(lm,n),e_k}), \overrightarrow{getfn(lm,rem(n,rem(m,ln)))}, \overrightarrow{le}) \rhd \delta^{\text{c}}\mathbf{0}\\
&+\sum_{(k,l)\in J\gamma J}\sum_{n:Nat}\sum_{m:Nat}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E_l}}\sum_{\overrightarrow{le:LE}} \gamma(\mathsf{a}_k,\mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}))^{\text{c}}t_k(\overrightarrow{getf1(lm,n),e_k})\\
&\quad\cdot \mathsf{Z}(t_k(\overrightarrow{getf1(lm,n),e_k}), remf2(lm,n,m))\\
&\qquad \lhd\, n < m \wedge m < lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})\\
&\qquad\ \wedge c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l}) \wedge remf2(lm,n,m) \neq \langle\rangle\\
&\qquad\ \wedge t \leq t_k(\overrightarrow{getf1(lm,n),e_k}) \wedge t_k(\overrightarrow{getf1(lm,n),e_k}) = t_l(\overrightarrow{getf1(lm,m),e_l})\\
&\qquad\ \wedge len(\overrightarrow{le}) + 2 = lenf(lm)\\
&\qquad\ \wedge sat\_mmd(t_k(\overrightarrow{getf1(lm,n),e_k}), \overrightarrow{getfn(lm,rem(n,rem(m,ln)))}, \overrightarrow{le}) \rhd \delta^{\text{c}}\mathbf{0}\\
&+\sum_{(k,l)\in J\gamma J}\sum_{n:Nat}\sum_{m:Nat}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E_l}} \gamma(\mathsf{a}_k,\mathsf{a}_l)(\overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}))^{\text{c}}t_k(\overrightarrow{getf1(lm,n),e_k})\\
&\qquad \lhd\, n < m \wedge m < lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1(lm,m),e'_l})\\
&\qquad\ \wedge c_k(\overrightarrow{getf1(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1(lm,m),e'_l}) \wedge remf2(lm,n,m) = \langle\rangle\\
&\qquad\ \wedge t \leq t_k(\overrightarrow{getf1(lm,n),e_k}) \wedge t_k(\overrightarrow{getf1(lm,n),e_k}) = t_l(\overrightarrow{getf1(lm,m),e_l}) \rhd \delta^{\text{c}}\mathbf{0}
\end{aligned}
$$

where $P\gamma Q = \{(k,l) \in P \times Q \mid \gamma(\mathsf{a}_k,\mathsf{a}_l) \text{ is defined}\}$.

Conjecture 6.3.1. $(\mathsf{X}(\overrightarrow{t}), G_7) \overset{ind}{\Rightarrow} (\mathsf{Z}(\mathbf{0}, seq1(\overrightarrow{t}, LM0)), L)$.

The other three cases considered in Section 5.2 can be scaled to the timed μCRL case in a similar way.

6.3.3 Relation between TLPEs and LPEs

In this section we explain how a *time-free abstraction* (cf. [55, Section 4.2]) of a TLPE can be obtained. First we prove some facts about the TLPE Z from the previous section.

Proposition 6.3.2. *The equation for* Z *is well-timed.*

Proof. It can be easily checked that $t \gg \mathsf{Z}(t, lm) \approx \mathsf{Z}(t, lm)$. The statement of the proposition follows from the fact that after performing an action $\mathsf{a}_i(\overrightarrow{t}) \,{}^{\varsigma}\, t$, the process behaves as $\mathsf{Z}(t, lm')$. □

Definition 6.3.3 (Deadlock-saturation). A TLPE X is *deadlock-saturated* if it is defined in the following way:

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) \,{}^{\varsigma}\, t_i(\overrightarrow{d,e_i}) \cdot \mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d,e_i})) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0}\\
&+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \,{}^{\varsigma}\, t_j(\overrightarrow{d,e_j}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta{}^{\varsigma}\mathbf{0}\\
&+\sum_{u:Time}\sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, u \lhd u \le t_\delta(\overrightarrow{d,e_\delta}) \wedge c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned} \tag{6.8}$$

Proposition 6.3.4. *The equation for* Z *is deadlock-saturated.*

Next, we use the embedding of well-timed basic terms into untimed μCRL terms defined in [55, Section 4.2]. For every action label $\mathsf{a} : \overrightarrow{D_\mathsf{a}} \to Proc$ in the given timed μCRL specification, we construct the action label $\bar{\mathsf{a}} : \overrightarrow{D_\mathsf{a}} \times Time \to Proc$. We also construct the action label $\Delta : \overrightarrow{Time} \to Proc$.

Definition 6.3.5. The *time-free abstraction* of TLPE X defined in (6.8) is the LPE Y defined as follows:

$$\begin{aligned}
\mathsf{Y}(\overrightarrow{d{:}D}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \overline{\mathsf{a}_i}(\overrightarrow{f_i}(\overrightarrow{d,e_i}), t_i(\overrightarrow{d,e_i})) \cdot \mathsf{Y}(\overrightarrow{g_i}(\overrightarrow{d,e_i})) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0}\\
&+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \overline{\mathsf{a}_j}(\overrightarrow{f_j}(\overrightarrow{d,e_j}), t_j(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta{}^{\varsigma}\mathbf{0}\\
&+\sum_{u:Time}\sum_{\overrightarrow{e_\delta:E_\delta}} \Delta(u) \lhd u \le t_\delta(\overrightarrow{d,e_\delta}) \wedge c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned} \tag{6.9}$$

The time-free abstraction of well-timed deadlock-saturated TLPEs can be used for further analysis with methods that are designed for untimed μCRL. For instance, strong bisimilarity of time-free abstractions of two well-timed deadlock-saturated TLPEs is equivalent to the timed bisimilarity of them. In the initial timed μCRL specification time has a direct influence on the specified behavior, for instance on the interleavings of parallel components (for example $\mathsf{a} \,{}^{\varsigma}\, 1 \parallel \mathsf{b} \,{}^{\varsigma}\, 2 \approx \mathsf{a} \,{}^{\varsigma}\, 1 \cdot \mathsf{b} \,{}^{\varsigma}\, 2$ in timed μCRL). This is why performing the time-free abstraction on the initial specification will not work (because $\mathsf{a}(1) \parallel \mathsf{b}(2) \not\approx \mathsf{a}(1) \cdot \mathsf{b}(2)$ in μCRL). However, after linearization the influence of time on the specified behavior is encoded in the parameters and conditions of resulting TLPE, i.e. time becomes just a conventional data type in untimed μCRL.

Appendix A

Final LPE Definitions

A.1 Final LPE for the Case with the Renaming Operations and Handshaking

$$
\begin{aligned}
&\mathsf{Z}(lm{:}ALM) = \\
&\quad \sum_{i\in I\setminus I_\tau} \sum_{\mathsf{a}\in\mathbf{R}(i)} \sum_{n:Nat} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}(\overrightarrow{f_i}(\overrightarrow{getf1d(lm,n),e_i})) \\
&\qquad \cdot \mathsf{Z}(replf1(lm,n,\mathbf{mklm}_i[p_i](\overrightarrow{getf1d(lm,n),e_i}))) \\
&\qquad \lhd n < lenf(lm) \land c_i(\overrightarrow{getf1d(lm,n),e_i}) \\
&\qquad\quad \land \mathbf{mka}[\mathsf{a}_i] \notin getH(getf1a(lm,n)) \cup getI(getf1a(lm,n)) \\
&\qquad\quad \land \mathbf{mka}[\mathsf{a}] = appl(\mathbf{mka}[\mathsf{a}_i], getR(getf1a(lm,n))) \rhd \delta \\
&+ \sum_{i\in I\setminus I_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_i:E_i}} \tau \cdot \mathsf{Z}(replf1(lm,n,\mathbf{mklm}_i[p_i](\overrightarrow{getf1d(lm,n),e_i}))) \\
&\qquad \lhd n < lenf(lm) \land c_i(\overrightarrow{getf1d(lm,n),e_i}) \\
&\qquad\quad \land \mathbf{mka}[\mathsf{a}_i] \in getI(getf1a(lm,n)) \setminus getH(getf1a(lm,n)) \rhd \delta \\
&+ \sum_{i\in I_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_i:E_i}} \tau \cdot \mathsf{Z}(replf1(lm,n,\mathbf{mklm}_i[p_i](\overrightarrow{getf1d(lm,n),e_i}))) \\
&\qquad \lhd n < lenf(lm) \land c_i(\overrightarrow{getf1d(lm,n),e_i}) \rhd \delta \\
&+ \sum_{j\in J\setminus J_\tau} \sum_{\mathsf{a}\in\mathbf{R}(j)} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}(\overrightarrow{f_j}(\overrightarrow{getf1d(lm,n),e_j})) \cdot \mathsf{Z}(remf1(lm,n)) \\
&\qquad \lhd n < lenf(lm) \land remf1(lm,n) \neq \langle\rangle \land c_j(\overrightarrow{getf1d(lm,n),e_j}) \\
&\qquad\quad \land \mathbf{mka}[\mathsf{a}_j] \notin getH(getf1a(lm,n)) \cup getI(getf1a(lm,n)) \\
&\qquad\quad \land \mathbf{mka}[\mathsf{a}] = appl(\mathbf{mka}[\mathsf{a}_j], getR(getf1a(lm,n))) \rhd \delta
\end{aligned}
$$

$$+ \sum_{j \in J \backslash J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \cdot \mathsf{Z}(\mathit{remf1}(lm,n))$$

$$\lhd\, n < \mathit{lenf}(lm) \wedge \mathit{remf1}(lm,n) \neq \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1d}(lm,n), e_j})$$
$$\wedge\, \mathbf{mka}[\mathsf{a}_j] \in \mathit{getI}(\mathit{getf1a}(lm,n)) \setminus \mathit{getH}(\mathit{getf1a}(lm,n)) \rhd \delta$$

$$+ \sum_{j \in J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \cdot \mathsf{Z}(\mathit{remf1}(lm,n))$$

$$\lhd\, n < \mathit{lenf}(lm) \wedge \mathit{remf1}(lm,n) \neq \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1d}(lm,n), e_j}) \rhd \delta$$

$$+ \sum_{j \in J \backslash J_\tau} \sum_{\mathsf{a} \in \mathbf{R}(j)} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}(\overrightarrow{f_j}(\overrightarrow{\mathit{getf1d}(lm,n), e_j}))$$

$$\lhd\, n < \mathit{lenf}(lm) \wedge \mathit{remf1}(lm,n) = \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1d}(lm,n), e_j})$$
$$\wedge\, \mathbf{mka}[\mathsf{a}_j] \notin \mathit{getH}(\mathit{getf1a}(lm,n)) \cup \mathit{getI}(\mathit{getf1a}(lm,n))$$
$$\wedge\, \mathbf{mka}[\mathsf{a}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_j], \mathit{getR}(\mathit{getf1a}(lm,n))) \rhd \delta$$

$$+ \sum_{j \in J \backslash J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \lhd n < \mathit{lenf}(lm) \wedge \mathit{remf1}(lm,n) = \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1d}(lm,n), e_j})$$
$$\wedge\, \mathbf{mka}[\mathsf{a}_j] \in \mathit{getI}(\mathit{getf1a}(lm,n)) \setminus \mathit{getH}(\mathit{getf1a}(lm,n)) \rhd \delta$$

$$+ \sum_{j \in J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \lhd n < \mathit{lenf}(lm) \wedge \mathit{remf1}(lm,n) = \langle\rangle \wedge c_j(\overrightarrow{\mathit{getf1d}(lm,n), e_j}) \rhd \delta$$

$$+ \sum_{(k,l) \in (I \backslash I_\tau)^2} \sum_{(\mathsf{a},\mathsf{b},\mathsf{c}) \in \mathbf{R}^{\mathbf{3}}_\gamma(k,l)} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \mathsf{a}(\overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}))$$

$$\cdot\, \mathsf{Z}(\mathit{replf2}(lm,n,m,\mathbf{mklm}_k[p_k](\overrightarrow{\mathit{getf1d}(lm,n), e_k}), \mathbf{mklm}_l[p_l](\overrightarrow{\mathit{getf1d}(lm,m), e'_l})))$$

$$\lhd\, n < m \wedge m < \mathit{lenf}(lm) \wedge \overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$
$$\wedge\, c_k(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) \wedge c_l(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$
$$\wedge\, \mathbf{mka}[\mathsf{a}_k] \notin \mathit{getH}(\mathit{getf2a0}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a0}(lm,n,m))$$
$$\wedge\, \mathbf{mka}[\mathsf{a}_l] \notin \mathit{getH}(\mathit{getf2a1}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a1}(lm,n,m))$$
$$\wedge\, \mathbf{mka}[\mathsf{b}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_k], \mathit{getR}(\mathit{getf2a0}(lm,n,m)))$$
$$\wedge\, \mathbf{mka}[\mathsf{c}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_l], \mathit{getR}(\mathit{getf2a1}(lm,n,m)))$$
$$\wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})] \notin \mathit{getH}(\mathit{getf2a}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a}(lm,n,m))$$
$$\wedge\, \mathbf{mka}[\mathsf{a}] = \mathit{appl}(\mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})], \mathit{getR}(\mathit{getf2a}(lm,n,m))) \rhd \delta$$

$$+ \sum_{(k,l)\in(I\backslash I_\tau)^2} \sum_{(\mathsf{b},\mathsf{c})\in\mathbf{R}^2_\gamma(k,l)} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \tau$$

$$\cdot \mathsf{Z}(\mathit{replf2}(lm, n, m, \mathbf{mklm}_k[p_k](\overrightarrow{\mathit{getf1d}(lm,n), e_k}), \mathbf{mklm}_l[p_l](\overrightarrow{\mathit{getf1d}(lm,m), e'_l})))$$

$$\lhd\, n < m \wedge m < \mathit{lenf}(lm) \wedge \overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, c_k(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) \wedge c_l(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_k] \notin \mathit{getH}(\mathit{getf2a0}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a0}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_l] \notin \mathit{getH}(\mathit{getf2a1}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a1}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{b}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_k], \mathit{getR}(\mathit{getf2a0}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\mathsf{c}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_l], \mathit{getR}(\mathit{getf2a1}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})] \in \mathit{getH}(\mathit{getf2a}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a}(lm,n,m)) \rhd \delta$$

$$+ \sum_{(k,l)\in(I\backslash I_\tau)\times(J\backslash J_\tau)} \sum_{(\mathsf{a},\mathsf{b},\mathsf{c})\in\mathbf{R}^3_\gamma(k,l)} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \mathsf{a}(\overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}))$$

$$\cdot \mathsf{Z}(\mathit{replremf2}(lm, n, m, \mathbf{mklm}_k[p_k](\overrightarrow{\mathit{getf1d}(lm,n), e_k})))$$

$$\lhd\, n \neq m \wedge n < \mathit{lenf}(lm) \wedge m < \mathit{lenf}(lm)$$

$$\wedge\, \overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, c_k(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) \wedge c_l(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_k] \notin \mathit{getH}(\mathit{getf2a0}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a0}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_l] \notin \mathit{getH}(\mathit{getf2a1}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a1}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{b}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_k], \mathit{getR}(\mathit{getf2a0}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\mathsf{c}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_l], \mathit{getR}(\mathit{getf2a1}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})] \notin \mathit{getH}(\mathit{getf2a}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{a}] = \mathit{appl}(\mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})], \mathit{getR}(\mathit{getf2a}(lm,n,m))) \rhd \delta$$

$$+ \sum_{(k,l)\in(I\backslash I_\tau)\times(J\backslash J_\tau)} \sum_{(\mathsf{b},\mathsf{c})\in\mathbf{R}^2_\gamma(k,l)} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \tau$$

$$\cdot \mathsf{Z}(\mathit{replremf2}(lm, n, m, \mathbf{mklm}_k[p_k](\overrightarrow{\mathit{getf1d}(lm,n), e_k})))$$

$$\lhd\, n \neq m \wedge n < \mathit{lenf}(lm) \wedge m < \mathit{lenf}(lm)$$

$$\wedge\, \overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, c_k(\overrightarrow{\mathit{getf1d}(lm,n), e_k}) \wedge c_l(\overrightarrow{\mathit{getf1d}(lm,m), e'_l})$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_k] \notin \mathit{getH}(\mathit{getf2a0}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a0}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{a}_l] \notin \mathit{getH}(\mathit{getf2a1}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a1}(lm,n,m))$$

$$\wedge\, \mathbf{mka}[\mathsf{b}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_k], \mathit{getR}(\mathit{getf2a0}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\mathsf{c}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_l], \mathit{getR}(\mathit{getf2a1}(lm,n,m)))$$

$$\wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})] \in \mathit{getH}(\mathit{getf2a}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a}(lm,n,m)) \rhd \delta$$

$$
\begin{aligned}
&+\sum_{(k,l)\in(J\backslash J_\tau)^2}\ \sum_{(\mathsf{a},\mathsf{b},\mathsf{c})\in\mathbf{R}^3_\gamma(k,l)}\ \sum_{n:Nat}\ \sum_{m:Nat}\ \sum_{\overrightarrow{e_k:E_k}}\ \sum_{\overrightarrow{e'_l:E_l}} \mathsf{a}(\overrightarrow{getf1d(lm,n),e_k})\cdot \mathsf{Z}(remf2(lm,n,m))\\
&\qquad \lhd\, n<m \wedge m<lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1d(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1d(lm,m),e'_l})\\
&\qquad \wedge\, c_k(\overrightarrow{getf1d(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1d(lm,m),e'_l}) \wedge remf2(lm,n,m)\neq\langle\,\rangle\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_k]\notin getH(getf2a0(lm,n,m))\cup getI(getf2a0(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_l]\notin getH(getf2a1(lm,n,m))\cup getI(getf2a1(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{b}] = appl(\mathbf{mka}[\mathsf{a}_k], getR(getf2a0(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{c}] = appl(\mathbf{mka}[\mathsf{a}_l], getR(getf2a1(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})]\notin getH(getf2a(lm,n,m))\cup getI(getf2a(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}] = appl(\mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})], getR(getf2a(lm,n,m)))\rhd\delta\\
&+\sum_{(k,l)\in(J\backslash J_\tau)^2}\ \sum_{(\mathsf{b},\mathsf{c})\in\mathbf{R}^2_\gamma(k,l)}\ \sum_{n:Nat}\ \sum_{m:Nat}\ \sum_{\overrightarrow{e_k:E_k}}\ \sum_{\overrightarrow{e'_l:E_l}} \tau\cdot \mathsf{Z}(remf2(lm,n,m))\\
&\qquad \lhd\, n<m \wedge m<lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1d(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1d(lm,m),e'_l})\\
&\qquad \wedge\, c_k(\overrightarrow{getf1d(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1d(lm,m),e'_l}) \wedge remf2(lm,n,m)\neq\langle\,\rangle\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_k]\notin getH(getf2a0(lm,n,m))\cup getI(getf2a0(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_l]\notin getH(getf2a1(lm,n,m))\cup getI(getf2a1(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{b}] = appl(\mathbf{mka}[\mathsf{a}_k], getR(getf2a0(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{c}] = appl(\mathbf{mka}[\mathsf{a}_l], getR(getf2a1(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})]\in getH(getf2a(lm,n,m))\cup getI(getf2a(lm,n,m))\rhd\delta\\
&+\sum_{(k,l)\in(J\backslash J_\tau)^2}\ \sum_{(\mathsf{a},\mathsf{b},\mathsf{c})\in\mathbf{R}^3_\gamma(k,l)}\ \sum_{n:Nat}\ \sum_{m:Nat}\ \sum_{\overrightarrow{e_k:E_k}}\ \sum_{\overrightarrow{e'_l:E_l}} \mathsf{a}(\overrightarrow{f_k}(\overrightarrow{getf1d(lm,n),e_k}))\\
&\qquad \lhd\, n<m \wedge m<lenf(lm) \wedge \overrightarrow{f_k}(\overrightarrow{getf1d(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{getf1d(lm,m),e'_l})\\
&\qquad \wedge\, c_k(\overrightarrow{getf1d(lm,n),e_k}) \wedge c_l(\overrightarrow{getf1d(lm,m),e'_l}) \wedge remf2(lm,n,m)=\langle\,\rangle\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_k]\notin getH(getf2a0(lm,n,m))\cup getI(getf2a0(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}_l]\notin getH(getf2a1(lm,n,m))\cup getI(getf2a1(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{b}] = appl(\mathbf{mka}[\mathsf{a}_k], getR(getf2a0(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{c}] = appl(\mathbf{mka}[\mathsf{a}_l], getR(getf2a1(lm,n,m)))\\
&\qquad \wedge\, \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})]\notin getH(getf2a(lm,n,m))\cup getI(getf2a(lm,n,m))\\
&\qquad \wedge\, \mathbf{mka}[\mathsf{a}] = appl(\mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})], getR(getf2a(lm,n,m)))\rhd\delta
\end{aligned}
$$

$$+ \sum_{(k,l)\in(J\setminus J_\tau)^2} \sum_{(\mathsf{b},\mathsf{c})\in\mathbf{R}^2_\gamma(k,l)} \sum_{n:Nat} \sum_{m:Nat} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E_l}} \tau$$

$$\begin{aligned}
&\lhd\, n < m \wedge m < \mathit{lenf}(lm) \wedge \overrightarrow{f_k}(\overrightarrow{\mathit{getf1d}(lm,n),e_k}) = \overrightarrow{f_l}(\overrightarrow{\mathit{getf1d}(lm,m),e'_l}) \\
&\wedge c_k(\overrightarrow{\mathit{getf1d}(lm,n),e_k}) \wedge c_l(\overrightarrow{\mathit{getf1d}(lm,m),e'_l}) \wedge \mathit{remf2}(lm,n,m) = \langle\,\rangle \\
&\wedge \mathbf{mka}[\mathsf{a}_k] \notin \mathit{getH}(\mathit{getf2a0}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a0}(lm,n,m)) \\
&\wedge \mathbf{mka}[\mathsf{a}_l] \notin \mathit{getH}(\mathit{getf2a1}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a1}(lm,n,m)) \\
&\wedge \mathbf{mka}[\mathsf{b}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_k], \mathit{getR}(\mathit{getf2a0}(lm,n,m))) \\
&\wedge \mathbf{mka}[\mathsf{c}] = \mathit{appl}(\mathbf{mka}[\mathsf{a}_l], \mathit{getR}(\mathit{getf2a1}(lm,n,m))) \\
&\wedge \mathbf{mka}[\gamma(\mathsf{b},\mathsf{c})] \in \mathit{getH}(\mathit{getf2a}(lm,n,m)) \cup \mathit{getI}(\mathit{getf2a}(lm,n,m)) \rhd \delta
\end{aligned}$$

where

$$\begin{aligned}
&I_\tau = \{i \in I \mid \mathsf{a}_i = \tau\} \qquad J_\tau = \{j \in J \mid \mathsf{a}_j = \tau\} \\
&\mathbf{R}(i) = \{\mathsf{a} \in \mathit{ActLab} \mid \mathit{type}(\mathsf{a}) = \mathit{type}(\mathsf{a}_i)\} \\
&\mathbf{R}^2_\gamma(k,l) = \{(\mathsf{b},\mathsf{c}) \in \mathit{ActLab}^2 \mid \mathit{type}(\mathsf{b}) = \mathit{type}(\mathsf{a}_k) = \mathit{type}(\mathsf{c}) = \mathit{type}(\mathsf{a}_l) \\
&\qquad\qquad \wedge \gamma(\mathsf{a},\mathsf{b}) \text{ is defined}\} \\
&\mathbf{R}^3_\gamma(k,l) = \{(\mathsf{a},\mathsf{b},\mathsf{c}) \in \mathit{ActLab} \times \mathbf{R}^2_\gamma(k,l) \mid \mathit{type}(\mathsf{a}) = \mathit{type}(\mathsf{b})\}
\end{aligned}$$

The LPE Z is in a sense an extension of the LPE we obtained for the case without the remaining operations. The first nine summands correspond to the first three summands of the latter LPE, so each of the interleaving possibilities is represented by three summands. The first one represents the case when the action (not τ) is not encapsulated or hidden, but can be renamed. The second one represents the case when the action (not τ) is not encapsulated, but hidden. And the third one represents the τ summands (we treat them separately, because τ cannot be encapsulated, hidden or renamed). There is no summand for the encapsulated actions, as they all become equal to δ and vanish.

In the case of handshakings, we get only two summands for each summand in the case without the renaming operations. This is because τ does not communicate and we do not need an additional summand for it.

A.2 Final LPE for the Case with Renaming and Multi-Party Communication

Without loss of generality, we assume that $J \setminus J_\tau = \{0, \ldots, k\}$ and $I \setminus I_\tau = \{k+1, \ldots, m\}$.

$$
\begin{array}{l}
\mathsf{Z}(lm{:}ALM) = \\
\quad \sum\limits_{i \in I \setminus I_\tau} \sum\limits_{\mathsf{a} \in \mathbf{R}(i)} \sum\limits_{ln_0 : LNat} \cdots \sum\limits_{ln_m : LNat} \sum\limits_{\overrightarrow{le_0 : LE_0}} \cdots \sum\limits_{\overrightarrow{le_m : LE_m}} \mathsf{a}(\overrightarrow{f0}(lm, ln_0, \ldots, ln_m, \\
\qquad head(\overrightarrow{le_0}), \ldots, head(\overrightarrow{le_m}))) \cdot \mathsf{Z}(replfn(lm, ln, \\
\qquad \mathbf{cat}[\mathbf{mkllm}[p_0](\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, \mathbf{mkllm}[p_m](\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})])) \\
\qquad \triangleleft\, lnI \neq LNat0 \wedge len(ln) \leq lenf(lm) \wedge is\_unique(ln) \wedge \bigwedge_{0 \leq l \leq m} is\_sorted(ln_l) \\
\qquad\quad \wedge \bigwedge_{0 \leq l \leq m} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge_{0 \leq l \leq m} len(ln_l) = len(\overrightarrow{le_l}) \\
\qquad\quad \wedge\, EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})]) \\
\qquad\quad \wedge\, C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m}) \\
\qquad\quad \wedge\, is\_act(\mathbf{mka}[\mathsf{a}], lm, ln, \\
\qquad\qquad \mathbf{cat}[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_m), \mathbf{mka}[\mathsf{a}_m])]) \triangleright \delta \\
\quad + \sum\limits_{i \in I \setminus I_\tau} \sum\limits_{ln_0 : LNat} \cdots \sum\limits_{ln_m : LNat} \sum\limits_{\overrightarrow{le_0 : LE_0}} \cdots \sum\limits_{\overrightarrow{le_m : LE_m}} \tau \cdot \mathsf{Z}(replfn(lm, ln, \\
\qquad \mathbf{cat}[\mathbf{mkllm}[p_0](\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, \mathbf{mkllm}[p_m](\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})])) \\
\qquad \triangleleft\, lnI \neq LNat0 \wedge len(ln) \leq lenf(lm) \wedge is\_unique(ln) \wedge \bigwedge_{0 \leq l \leq m} is\_sorted(ln_l) \\
\qquad\quad \wedge \bigwedge_{0 \leq l \leq m} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge_{0 \leq l \leq m} len(ln_l) = len(\overrightarrow{le_l}) \\
\qquad\quad \wedge\, EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m})]) \\
\qquad\quad \wedge\, C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_m(\overrightarrow{getfn(lm, ln_m)}, \overrightarrow{le_m}) \\
\qquad\quad \wedge\, is\_tau(lm, ln, \\
\qquad\qquad mtcat[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_m), \mathbf{mka}[\mathsf{a}_m])]) \triangleright \delta \\
\quad + \sum\limits_{i \in I_\tau} \sum\limits_{n : Nat} \sum\limits_{\overrightarrow{e_i : E_i}} \tau \cdot \mathsf{Z}(replf1(lm, n, \mathbf{mklm}_i[p_i](\overrightarrow{getf1d(lm, n), e_i}))) \\
\qquad \triangleleft\, n < lenf(lm) \wedge c_i(\overrightarrow{getf1d(lm, n), e_i}) \triangleright \delta
\end{array}
$$

$$
\begin{aligned}
&+ \sum_{j\in J\setminus J_\tau} \sum_{\mathsf{a}\in\mathbf{R}(j)} \sum_{ln_0:LNat}\cdots\sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}}\cdots\sum_{\overrightarrow{le_k:LE_k}} \mathsf{a}(\overrightarrow{f0}(lm, ln_0, \ldots, ln_k,\\
&\quad head(\overrightarrow{le_0}), \ldots, head(\overrightarrow{le_k}))) \cdot \mathsf{Z}(remfn(lm, lnJ))\\
&\quad \lhd\, lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_sorted(ln_l)\\
&\qquad \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge_{0\leq l\leq k} len(ln_l) = len(\overrightarrow{le_l})\\
&\qquad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})])\\
&\qquad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})\\
&\qquad \wedge is\_act(\mathbf{mka}[\mathsf{a}], lm, lnJ,\\
&\qquad\quad \mathbf{cat}[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_k), \mathbf{mka}[\mathsf{a}_k])])\\
&\qquad \wedge remfn(lm, lnJ) \neq \langle\rangle \rhd \delta\\
&+ \sum_{j\in J\setminus J_\tau} \sum_{ln_0:LNat}\cdots\sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}}\cdots\sum_{\overrightarrow{le_k:LE_k}} \tau \cdot \mathsf{Z}(remfn(lm, lnJ))\\
&\quad \lhd\, lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_sorted(ln_l)\\
&\qquad \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge_{0\leq l\leq k} len(ln_l) = len(\overrightarrow{le_l})\\
&\qquad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})])\\
&\qquad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})\\
&\qquad \wedge is\_tau(lm, lnJ,\\
&\qquad\quad \mathbf{cat}[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_k), \mathbf{mka}[\mathsf{a}_k])])\\
&\qquad \wedge remfn(lm, lnJ) \neq \langle\rangle \rhd \delta\\
&+ \sum_{j\in J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \cdot \mathsf{Z}(remf1(lm, n))\\
&\quad \lhd\, n < lenf(lm) \wedge remf1(lm, n) \neq \langle\rangle \wedge c_j(\overrightarrow{getf1d(lm, n), e_j}) \rhd \delta\\
&+ \sum_{j\in J\setminus J_\tau} \sum_{\mathsf{a}\in\mathbf{R}(j)} \sum_{ln_0:LNat}\cdots\sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}}\cdots\sum_{\overrightarrow{le_k:LE_k}} \mathsf{a}(\overrightarrow{f0}(lm, ln_0, \ldots, ln_k,\\
&\quad head(\overrightarrow{le_0}), \ldots, head(\overrightarrow{le_k})))\\
&\quad \lhd\, lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_sorted(ln_l)\\
&\qquad \wedge \textstyle\bigwedge_{0\leq l\leq k} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge_{0\leq l\leq k} len(ln_l) = len(\overrightarrow{le_l})\\
&\qquad \wedge EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})])\\
&\qquad \wedge C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})\\
&\qquad \wedge is\_act(\mathbf{mka}[\mathsf{a}], lm, lnJ,\\
&\qquad\quad \mathbf{cat}[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_k), \mathbf{mka}[\mathsf{a}_k])])\\
&\qquad \wedge remfn(lm, lnJ) = \langle\rangle \rhd \delta
\end{aligned}
$$

$$+ \sum_{j \in J \setminus J_\tau} \sum_{ln_0:LNat} \cdots \sum_{ln_k:LNat} \sum_{\overrightarrow{le_0:LE_0}} \cdots \sum_{\overrightarrow{le_k:LE_k}} \tau$$

$$\lhd\ lnJ \neq LNat0 \wedge len(lnJ) \leq lenf(lm) \wedge is\_unique(lnJ) \wedge \bigwedge\nolimits_{0 \leq l \leq k} is\_sorted(ln_l)$$
$$\wedge \bigwedge\nolimits_{0 \leq l \leq k} is\_each\_lower(lenf(lm), ln_l) \wedge \bigwedge\nolimits_{0 \leq l \leq k} len(ln_l) = len(\overrightarrow{le_l})$$
$$\wedge\ EQ(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})])$$
$$\wedge\ C_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}) \wedge \cdots \wedge C_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})$$
$$\wedge\ is\_tau(lm, lnJ,$$
$$\mathbf{cat}[mklact(len(ln_0), \mathbf{mka}[\mathsf{a}_0]), \ldots, mklact(len(ln_k), \mathbf{mka}[\mathsf{a}_k])])$$
$$\wedge\ remfn(lm, lnJ) = \langle\,\rangle \rhd \delta$$

$$+ \sum_{j \in J_\tau} \sum_{n:Nat} \sum_{\overrightarrow{e_j:E_j}} \tau \lhd n < lenf(lm) \wedge remf1(lm, n) = \langle\,\rangle \wedge c_j(\overrightarrow{getf1d(lm, n), e_j}) \rhd \delta$$

where $lnI = \mathbf{cat}[ln_{k+1}, \ldots, ln_m]$, $lnJ = \mathbf{cat}[ln_0, \ldots, ln_k]$, and $ln = cat(lnJ, lnI)$.

The first three sets of summands represent multi-party communications of several components with at least one of them not terminating. In the third set we separate the actions a_i that are equal to τ – they cannot communicate and can only be executed in the interleaving way. In the first set of summands we consider all non τ actions a_i and all possible renamings of them. We do not need to consider the renamings of actions a_j here because at least one of the components will be executing an a_i action, and therefore the resulting action will be a renaming of it.

As in the case of multi-party communications without renaming, we take a number of lists to identify which first layer elements will communicate by performing which actions. The condition $lnI \neq LNat0$ ensures that at least one of the elements will not terminate. Instead of checking the conformance to a chosen configuration, we use the function *is_act* to see if the result of the multi-party communication is the chosen action. The rest of the conditions are the same as in the case without renaming operations. The second set of summands is similar to the first one and captures the case when communication results in τ.

The following six summands capture the case when all components terminate after performing a communication. The first three represent the sub-case when the LPE Z does not terminate in such a situation, and the last three represent the sub-case when the LPE Z terminates.

In case the LPE Z performs an action, its parameters are the parameters of any of the communicating actions, so we take the first one. We could skip the definition of the function $\overrightarrow{f0}$ and use the following expression instead:

$$head(\mathbf{cat}[F_0(\overrightarrow{getfn(lm, ln_0)}, \overrightarrow{le_0}), \ldots, F_k(\overrightarrow{getfn(lm, ln_k)}, \overrightarrow{le_k})])$$

which, however, is a more complex expression.

Appendix B

Axioms and Proofs in Timed μCRL

B.1 Axioms of Timed μCRL

$x + y \approx y + x$	(A1)
$x + (y + z) \approx (x + y) + z$	(A2)
$x + x \approx x$	(A3)
$(x + y) \cdot z \approx x \cdot z + y \cdot z$	(A4)
$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$	(A5)
$x + \partial\mathcal{U}(x) \approx x$	(A6T)
$\delta + \partial\mathcal{U}(x) \approx \delta$	(A6T$'$)
$\delta \cdot x \approx \delta$	(A7)

Table B.1: Basic axioms of timed μCRL.

B.2 Derivable Identities in Timed μCRL

B.2.1 Derivable Identities of Sort *Time*

Below we list some useful identities that are derivable from the axioms of sort *Time*. These are properties of $\mathbf{0}$, $\leq$ (derivable from the total order axioms); properties of *eq* as an equivalence relation, and its connection with *if*; properties of *if*; and properties of *min* and *max* as a distributive lattice.

Lemma B.2.1. *The following identities are derivable from the axioms of Time:*

1. $t \leq \mathbf{0} \approx eq(t, \mathbf{0})$;
2. $t \leq t \approx \mathbf{t}$;

$x \parallel y \approx (x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x) + x \mid y$	(CM1)
$b \,^{c}\, t \mathbin{⌊\!⌊} y \approx (b \,^{c}\, t \ll y) \cdot y$	(CM2T)
$(b \,^{c}\, t \cdot x) \mathbin{⌊\!⌊} y \approx (b \,^{c}\, t \ll y) \cdot ((t \gg x) \parallel y)$	(CM3T)
$(x + y) \mathbin{⌊\!⌊} z \approx x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z$	(CM4)
$(b \cdot x) \mid b' \approx (b \mid b') \cdot x$	(CM5)
$(b \cdot x) \mid (b' \cdot y) \approx (b \mid b') \cdot (x \parallel y)$	(CM7)
$(x + y) \mid z \approx x \mid z + y \mid z$	(CM8)
$(x \mid y) \,^{c}\, t \approx x \,^{c}\, t \mid y$	(ATA7)
$(x \mid y) \,^{c}\, t \approx x \mid y \,^{c}\, t$	(ATA8)
$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) \approx \gamma(\mathsf{a},\mathsf{a}')(\overrightarrow{d}) \lhd \overrightarrow{d} = \overrightarrow{d'} \rhd \delta$ if $\gamma(\mathsf{a},\mathsf{a}')$ is defined	(CF1)
$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) \approx \delta$ otherwise	(CF2)
$\tau \mid b \approx \delta$	(CT1)
$x \mid y \approx y \mid x$	(SC3)

Table B.2: Axioms for parallel composition in timed μCRL.

$x \lhd \mathbf{t} \rhd y \approx x$	(Cond1)
$x \lhd \mathbf{f} \rhd y \approx y$	(Cond2)
$x \lhd c \rhd y \approx x \lhd c \rhd \delta \,^{c}\, \mathbf{0} + y \lhd \neg c \rhd \delta \,^{c}\, \mathbf{0}$	(Cond3T)
$(x \lhd c_1 \rhd \delta \,^{c}\, \mathbf{0}) \lhd c_2 \rhd \delta \,^{c}\, \mathbf{0} \approx (x \lhd c_1 \wedge c_2 \rhd \delta \,^{c}\, \mathbf{0})$	(Cond4T)
$(x \lhd c_1 \rhd \delta \,^{c}\, \mathbf{0}) + (x \lhd c_2 \rhd \delta \,^{c}\, \mathbf{0}) \approx x \lhd c_1 \vee c_2 \rhd \delta \,^{c}\, \mathbf{0}$	(Cond5T)
$(x \lhd c \rhd \delta \,^{c}\, \mathbf{0}) \cdot y \approx (x \cdot y) \lhd c \rhd \delta \,^{c}\, \mathbf{0}$	(Cond6T)
$(x + y) \lhd c \rhd \delta \,^{c}\, \mathbf{0} \approx x \lhd c \rhd \delta \,^{c}\, \mathbf{0} + y \lhd c \rhd \delta \,^{c}\, \mathbf{0}$	(Cond7T)
$(x \lhd c \rhd \delta \,^{c}\, \mathbf{0}) \mathbin{⌊\!⌊} y \approx (x \mathbin{⌊\!⌊} y) \lhd c \rhd \delta \,^{c}\, \mathbf{0}$	(Cond8T)
$(x \lhd c \rhd \delta \,^{c}\, \mathbf{0}) \mid y \approx (x \mid y) \lhd c \rhd \delta \,^{c}\, \mathbf{0}$	(Cond9T)
$(x \lhd c \rhd \delta \,^{c}\, \mathbf{0}) \cdot (y \lhd c \rhd \delta \,^{c}\, \mathbf{0}) \approx (x \cdot y) \lhd c \rhd \delta \,^{c}\, \mathbf{0}$	(ScaT)
$p \lhd eq(d,e) \rhd \delta \,^{c}\, \mathbf{0} \approx p[d := e] \lhd eq(d,e) \rhd \delta \,^{c}\, \mathbf{0}$	(PET)

Table B.3: Axioms for conditions in timed μCRL.

3. $eq(t,t) \approx \mathbf{t}$;
4. $eq(t,u) \approx eq(u,t)$;
5. $eq(if(b,t,u),w) \approx (b \wedge eq(t,w)) \vee (\neg b \wedge eq(u,w))$;
6. $t \leq u \approx t \leq u \vee \neg u \leq t$;
7. $\neg t \leq u \approx \neg t \leq u \wedge u \leq t$;
8. $\neg\, u \leq t \wedge \neg\, w \leq u \approx \neg\, u \leq t \wedge \neg\, w \leq u \wedge \neg\, w \leq t$;
9. $t \leq u \vee u \leq w \approx t \leq u \vee u \leq w \vee t \leq w$;
10. $t \leq u \vee \neg\, w \leq u \approx t \leq u \vee \neg\, w \leq u \vee t \leq w$;

$$\sum_{d:D} x \approx x \tag{SUM1}$$

$$\sum_{e:D} r \approx \sum_{d:D} (r[e := d]) \tag{SUM2}$$

$$\sum_{d:D} p \approx \sum_{d:D} p + p \tag{SUM3}$$

$$\sum_{d:D} (p + q) \approx \sum_{d:D} p + \sum_{d:D} q \tag{SUM4}$$

$$\sum_{d:D} (p \cdot x) \approx (\sum_{d:D} p) \cdot x \tag{SUM5}$$

$$\sum_{d:D} (p \,⫝\, x) \approx (\sum_{d:D} p) \,⫝\, x \tag{SUM6}$$

$$\sum_{d:D} (p \mid x) \approx (\sum_{d:D} p) \mid x \tag{SUM7}$$

$$\sum_{d:D} (\partial_H(p)) \approx \partial_H(\sum_{d:D} p) \tag{SUM8}$$

$$\sum_{d:D} (\tau_I(p)) \approx \tau_I(\sum_{d:D} p) \tag{SUM9}$$

$$\sum_{d:D} (\rho_R(p)) \approx \rho_R(\sum_{d:D} p) \tag{SUM10}$$

$$\sum_{d:D} (p \lhd c \rhd \delta^{\text{c}}\mathbf{0}) \approx (\sum_{d:D} p) \lhd c \rhd \delta^{\text{c}}\mathbf{0} \tag{SUM12T}$$

Table B.4: Axioms for sums in timed μCRL.

11. $\neg\ u \leq t \vee u \leq w \approx \neg\ u \leq t \vee u \leq w \vee t \leq w$;

12. $eq(t, u) \wedge eq(u, w) \approx eq(t, u) \wedge eq(u, w) \wedge eq(t, w)$;

13. $eq(t, u) \wedge \neg eq(u, w) \approx eq(t, u) \wedge \neg eq(u, w) \wedge \neg eq(t, w)$;

14. $if(b, t, t) \approx t$;

15. $if(\mathbf{f}, t, u) \approx u$;

16. $if(b_1 \wedge b_2, t, u) \approx if(b_1, if(b_2, t, u), u)$;

17. $if(b_1, t, if(b_2, u, w)) \approx if(b_1, t, if(\neg b_1 \wedge b_2, u, w))$;

18. $if(b_1, if(b_2, u, w), t) \approx if(b_1, if(b_1 \wedge b_2, u, w), t)$;

19. $if(b, t, u) \approx if(b \vee eq(t, u), t, u)$;

20. $if(t \leq u, t, u) \approx if(u \leq t, u, t)$;

21. $min(\mathbf{0}, t) \approx \mathbf{0}$;

22. $max(\mathbf{0}, t) \approx t$;

23. $min(t, t) \approx t$;

$\partial_H(b) \approx b$ if $b = \tau$ or ($b = \mathsf{a}(\vec{d})$ and $\mathsf{a} \notin H$)	(D1)
$\partial_H(b) \approx \delta$ otherwise	(D2)
$\partial_H(x + y) \approx \partial_H(x) + \partial_H(y)$	(D3)
$\partial_H(x \cdot y) \approx \partial_H(x) \cdot \partial_H(y)$	(D4)
$\partial_H(x \lhd c \rhd \delta^{c}\mathbf{0}) \approx \partial_H(x) \lhd c \rhd \delta^{c}\mathbf{0}$	(D5T)
$\partial_H(x {}^{c}\, t) \approx \partial_H(x) {}^{c}\, t$	(D7)
$\tau_I(b) \approx b$ if $b = \delta$ or ($b = \mathsf{a}(\vec{d})$ and $\mathsf{a} \notin I$)	(T1)
$\tau_I(b) \approx \tau$ otherwise	(T2)
$\tau_I(x + y) \approx \tau_I(x) + \tau_I(y)$	(T3)
$\tau_I(x \cdot y) \approx \tau_I(x) \cdot \tau_I(y)$	(T4)
$\tau_I(x \lhd c \rhd \delta^{c}\mathbf{0}) \approx \tau_I(x) \lhd c \rhd \delta^{c}\mathbf{0}$	(T5T)
$\tau_I(x {}^{c}\, t) \approx \tau_I(x) {}^{c}\, t$	(T7)
$\rho_R(\delta) \approx \delta$	(RD)
$\rho_R(\tau) \approx \tau$	(RT)
$\rho_R(\mathsf{a}(\vec{d})) \approx R(\mathsf{a})(\vec{d})$	(R1)
$\rho_R(x + y) \approx \rho_R(x) + \rho_R(y)$	(R3)
$\rho_R(x \cdot y) \approx \rho_R(x) \cdot \rho_R(y)$	(R4)
$\rho_R(x \lhd c \rhd \delta^{c}\mathbf{0}) \approx \rho_R(x) \lhd c \rhd \delta^{c}\mathbf{0}$	(R5T)
$\rho_R(x {}^{c}\, t) \approx \rho_R(x) {}^{c}\, t$	(R7)

Table B.5: Axioms for renaming operators in μCRL.

$(x \mathbin{\|\!\lfloor} y) \mathbin{\|\!\lfloor} z \approx x \mathbin{\|\!\lfloor} (y \parallel z)$	(SC1)
$(x \mid y) \mid z \approx x \mid (y \mid z)$	(SC4)
$x \mid (y \mathbin{\|\!\lfloor} z) \approx (x \mid y) \mathbin{\|\!\lfloor} z$	(SC5)
$x \mathbin{\|\!\lfloor} \delta \approx x \cdot \delta$	(SCD1)
$x \mid \delta \approx \partial\mathcal{U}(x)$	(SCDT2)
$x {}^{c}\, t \mathbin{\|\!\lfloor} y \approx (x \mathbin{\|\!\lfloor} y) {}^{c}\, t$	(SCT1)
$(x {}^{c}\, t \mathbin{\|\!\lfloor} u \gg y) \lhd u \leq t \rhd \delta^{c}\mathbf{0} \approx (x {}^{c}\, t \mathbin{\|\!\lfloor} y) \lhd u \leq t \rhd \delta^{c}\mathbf{0}$	(SCT2)

Table B.6: Axioms for Standard Concurrency in μCRL.

24. $max(t, t) \approx t$;

25. $min(t, u) \approx min(u, t)$;

26. $max(t, u) \approx max(u, t)$;

27. $t \leq min(u, w) \approx t \leq u \wedge t \leq w$;

28. $t \leq max(u, w) \approx t \leq u \vee t \leq w$;

29. $min(u, w) \leq t \approx u \leq t \vee w \leq t$;

30. $max(u, w) \leq t \approx u \leq t \wedge w \leq t$;

$x \approx \sum_{t:Time} x \,^{c}\, t$	(AT1)
$b \,^{c}\, t \cdot y \approx b \,^{c}\, t \cdot t \gg y$	(AT2)
$x \,^{c}\, t \,^{c}\, u \approx x \,^{c}\, t \lhd t = u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, min(t,u)$	(ATA1′)
$(x+y) \,^{c}\, t \approx x \,^{c}\, t + y \,^{c}\, t$	(ATA2)
$(x \cdot y) \,^{c}\, t \approx x \,^{c}\, t \cdot y$	(ATA3)
$(\sum_{d:D} p) \,^{c}\, t \approx \sum_{d:D} p \,^{c}\, t$	(ATA4)
$(x \lhd c \rhd \delta^{c}\mathbf{0}) \,^{c}\, t \approx x \,^{c}\, t \lhd c \rhd \delta^{c}\mathbf{0}$	(ATA5′)
$t \gg x \approx t \ggg x + \delta \,^{c}\, t$	(ATB0)
$t \ggg x \approx \sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}$	(ATD0)
$x \ll b \approx x$	(ATC1′)
$x \ll (y+z) \approx x \ll y + x \ll z$	(ATC2)
$x \ll (y \cdot z) \approx x \ll y$	(ATC3)
$x \ll \sum_{d:D} p \approx \sum_{d:D} x \ll p$	(ATC4)
$x \ll (y \lhd c \rhd \delta^{c}\mathbf{0}) \approx (x \ll y) \lhd c \rhd \delta^{c}\mathbf{0} + x \,^{c}\, \mathbf{0}$	(ATC5′)
$x \ll (y \mathbin{\lfloor\!\lfloor} z) \approx x \ll (y \ll z)$	(ATC6)
$x \ll (y \mid z) \approx x \ll (y \ll z)$	(ATC7)
$x \ll (\partial_H(y)) \approx x \ll y$	(ATC8)
$x \ll (\tau_I(y)) \approx x \ll y$	(ATC9)
$x \ll (\rho_R(y)) \approx x \ll y$	(ATC10)
$x \ll (y \,^{c}\, t) \approx \sum_{u:Time} (x \ll y) \,^{c}\, u \lhd u \leq t \rhd \delta^{c}\mathbf{0}$	(ATC11)
$x \ll (y \ll z) \approx (x \ll y) \ll z$	(ATC12)
$x \ll (y \ll z) \approx x \ll (z \ll y)$	(ATC13)
$\delta \ll x \approx \partial\mathcal{U}(x)$	(ATCC0)
$x \,^{c}\, \mathbf{0} \ll y \approx x \,^{c}\, \mathbf{0}$	(ATCC1)
$x \cdot y \ll z \approx (x \ll z) \cdot y$	(ATCC3)
$x \,^{c}\, t \ll y \approx (x \ll y) \,^{c}\, t$	(ATCC7)

Table B.7: Axioms for timing operations.

31. $t \leq min(t,u) \approx t \leq u$;

32. $t \leq max(t,u) \approx \mathbf{t}$;

33. $min(t,u) \leq t \approx \mathbf{t}$;

34. $max(t,u) \leq t \approx u \leq t$;

35. $t \leq u \approx eq(t, min(t,u))$;

36. $t \leq u \approx eq(u, max(t,u))$;

37. $min(t, max(t, u)) \approx t$;

38. $max(t, min(t, u)) \approx t$;

39. $min(t, min(u, w)) \approx min(min(t, u), w)$;

40. $max(t, max(u, w)) \approx max(max(t, u), w)$;

41. $min(t, max(u, w)) \approx max(min(t, u), min(t, w))$;

42. $max(t, min(u, w)) \approx min(max(t, u), max(t, w))$.

Proof. 1. $t \leq \mathbf{0} \approx eq(t, \mathbf{0})$;

$$\begin{aligned} & eq(t, \mathbf{0}) \\ \text{(Time5)} & \approx t \leq \mathbf{0} \wedge \mathbf{0} \leq t \\ \text{(Time2)} & \approx t \leq \mathbf{0} \wedge \mathbf{t} \\ & \approx t \leq \mathbf{0} \end{aligned}$$

2. $t \leq t \approx \mathbf{t}$;

$$\begin{aligned} & \mathbf{t} \\ \text{(Time3)} & \approx t \leq t \vee t \leq t \\ & \approx t \leq t \end{aligned}$$

3. $eq(t, t) \approx \mathbf{t}$;

$$\begin{aligned} & eq(t, t) \\ \text{(Time5)} & \approx t \leq t \wedge t \leq t \\ & \approx t \leq t \\ (2) & \approx \mathbf{t} \end{aligned}$$

4. $eq(t, u) \approx eq(u, t)$;

$$\begin{aligned} & eq(t, u) \\ \text{(Time5)} & \approx t \leq u \wedge u \leq t \\ & \approx u \leq t \wedge t \leq u \\ \text{(Time5)} & \approx eq(u, t) \end{aligned}$$

5. $eq(if(b, t, u), w) \approx (b \wedge eq(t, w)) \vee (\neg b \wedge eq(u, w))$;

$$\begin{aligned} & eq(if(b, t, u), w) \\ \text{(Time5)} & \approx if(b, t, u) \leq w \wedge w \leq if(b, t, u) \\ \text{(Time12), (Time13)} & \approx ((b \wedge t \leq w) \vee (\neg b \wedge u \leq w)) \wedge ((b \wedge w \leq t) \vee (\neg b \wedge w \leq u)) \\ & \approx (b \wedge t \leq w \wedge w \leq t) \vee (\neg b \wedge u \leq w \wedge w \leq u) \\ & \approx (b \wedge eq(t, w)) \vee (\neg b \wedge eq(u, w)) \end{aligned}$$

6. $t \leq u \approx t \leq u \vee \neg\, u \leq t$;

$$\begin{aligned} & t \leq u \\ & \approx t \leq u \vee \mathbf{f} \\ \text{(Time3)} & \approx t \leq u \vee \neg(t \leq u \vee u \leq t) \\ & \approx t \leq u \vee (\neg\, t \leq u \wedge \neg\, u \leq t) \\ & \approx t \leq u \vee \neg\, u \leq t \end{aligned}$$

7. $\neg\, t \leq u \approx \neg\, t \leq u \wedge u \leq t$;

$$\begin{array}{rl} \neg\, t \leq u & \\ & \approx \neg\, t \leq u \wedge \mathbf{t} \\ \text{(Time3)} & \approx \neg\, t \leq u \wedge (t \leq u \vee u \leq t) \\ & \approx \neg\, t \leq u \wedge u \leq t \end{array}$$

8. $\neg\, u \leq t \wedge \neg\, w \leq u \approx \neg\, u \leq t \wedge \neg\, w \leq u \wedge \neg\, w \leq t$;

$$\begin{array}{rl} \neg\, u \leq t \wedge \neg\, w \leq u & \\ (7) & \approx \neg\, u \leq t \wedge \neg\, w \leq u \wedge u \leq w \\ \text{(Time1}'''\text{)} & \approx \neg\, u \leq t \wedge \neg\, w \leq u \wedge u \leq w \wedge \neg\, w \leq t \\ (7) & \approx \neg\, u \leq t \wedge \neg\, w \leq u \wedge \neg\, w \leq t \end{array}$$

9. $t \leq u \vee u \leq w \approx t \leq u \vee u \leq w \vee t \leq w$;

$$\begin{array}{rl} t \leq u \vee u \leq w & \\ & \approx \neg\neg(t \leq u \vee u \leq w) \\ & \approx \neg(\neg\, t \leq u \wedge \neg\, u \leq w) \\ (8) & \approx \neg(\neg\, t \leq u \wedge \neg\, u \leq w \wedge \neg\, t \leq w) \\ & \approx t \leq u \vee u \leq w \vee t \leq w \end{array}$$

10. $t \leq u \vee \neg\, w \leq u \approx t \leq u \vee \neg\, w \leq u \vee t \leq w$;

$$\begin{array}{rl} t \leq u \vee \neg\, w \leq u & \\ & \approx \neg\neg(t \leq u \vee \neg\, w \leq u) \\ & \approx \neg(\neg\, t \leq u \wedge w \leq u) \\ \text{(Time1}''\text{)} & \approx \neg(\neg\, t \leq u \wedge w \leq u \wedge \neg\, t \leq w) \\ & \approx t \leq u \vee \neg\, w \leq u \vee t \leq w \end{array}$$

11. $\neg\, u \leq t \vee u \leq w \approx \neg\, u \leq t \vee u \leq w \vee t \leq w$;

$$\begin{array}{rl} \neg\, u \leq t \vee u \leq w & \\ & \approx \neg\neg(\neg\, u \leq t \vee u \leq w) \\ & \approx \neg(\, u \leq t \wedge \neg\, u \leq w) \\ \text{(Time1}'''\text{)} & \approx \neg(\, u \leq t \wedge \neg\, u \leq w \wedge \neg\, t \leq w) \\ & \approx \neg\, u \leq t \vee u \leq w \vee t \leq w \end{array}$$

12. $eq(t,u) \wedge eq(u,w) \approx eq(t,u) \wedge eq(u,w) \wedge eq(t,w)$;

$$\begin{array}{rl} eq(t,u) \wedge eq(u,w) \wedge eq(t,w) & \\ \text{3 times (Time5)} & \approx t \leq u \wedge u \leq t \wedge u \leq w \wedge w \leq u \wedge t \leq w \wedge w \leq t \\ \text{twice (Time1}'\text{)} & \approx t \leq u \wedge u \leq t \wedge u \leq w \wedge w \leq u \\ \text{twice (Time5)} & \approx eq(t,u) \wedge eq(u,w) \end{array}$$

13. $eq(t,u) \wedge \neg eq(u,w) \approx eq(t,u) \wedge \neg eq(u,w) \wedge \neg eq(t,w)$;

$$
\begin{aligned}
& eq(t,u) \wedge \neg eq(u,w) \wedge \neg eq(t,w) \\
\text{3 times (Time5)} & \approx t \leq u \wedge u \leq t \wedge \neg(u \leq w \wedge w \leq u) \wedge \neg(t \leq w \wedge w \leq t) \\
& \approx t \leq u \wedge u \leq t \wedge (\neg\, u \leq w \vee \neg\, w \leq u) \wedge (\neg\, t \leq w \vee \neg\, w \leq t) \\
& \approx t \leq u \wedge u \leq t \wedge (\\
& \qquad (\neg\, u \leq w \wedge \neg\, t \leq w) \vee \\
& \qquad (\neg\, u \leq w \wedge \neg\, w \leq t) \vee \\
& \qquad (\neg\, w \leq u \wedge \neg\, t \leq w) \vee \\
& \qquad (\neg\, w \leq u \wedge \neg\, w \leq t)) \\
\text{twice (8)} & \approx t \leq u \wedge u \leq t \wedge (\\
& \qquad (\neg\, u \leq w \wedge \neg\, t \leq w) \vee \\
& \qquad (\neg\, u \leq w \wedge \neg\, w \leq t \wedge \neg\, u \leq t) \vee \\
& \qquad (\neg\, w \leq u \wedge \neg\, t \leq w \wedge \neg\, t \leq u) \vee \\
& \qquad (\neg\, w \leq u \wedge \neg\, w \leq t)) \\
& \approx (t \leq u \wedge u \leq t \wedge \neg\, u \leq w \wedge \neg\, t \leq w) \vee \\
& \qquad (t \leq u \wedge u \leq t \wedge \neg\, w \leq u \wedge \neg\, w \leq t) \\
\text{twice (Time1}'') & \approx (t \leq u \wedge u \leq t \wedge \neg\, u \leq w) \vee \\
& \qquad (t \leq u \wedge u \leq t \wedge \neg\, w \leq u) \\
& \approx (t \leq u \wedge u \leq t \wedge (\neg\, u \leq w \vee \neg\, w \leq u) \\
& \approx (t \leq u \wedge u \leq t \wedge \neg(u \leq w \wedge w \leq u) \\
\text{twice (Time5)} & \approx eq(t,u) \wedge \neg eq(u,w)
\end{aligned}
$$

14. $\mathit{if}(b,t,t) \approx t$;

$$
\begin{aligned}
& \mathit{if}(b,t,t) \\
\text{(Time7)} & \approx \mathit{if}(b,t,\mathit{if}(\mathbf{t},t,t)) \\
\text{(Time9)} & \approx \mathit{if}(b \vee \mathbf{t},t,t) \\
& \approx \mathit{if}(\mathbf{t},t,t) \\
\text{(Time7)} & \approx t
\end{aligned}
$$

15. $\mathit{if}(\mathbf{f},t,u) \approx u$;

$$
\begin{aligned}
& \mathit{if}(\mathbf{f},t,u) \\
& \approx \mathit{if}(\neg\mathbf{t},t,u) \\
\text{(Time8}') & \approx \mathit{if}(\mathbf{t},u,t) \\
\text{(Time7)} & \approx u
\end{aligned}
$$

16. $\mathit{if}(b_1 \wedge b_2,t,u) \approx \mathit{if}(b_1,\mathit{if}(b_2,t,u),u)$;

$$
\begin{aligned}
& \mathit{if}(b_1 \wedge b_2,t,u) \\
& \approx \mathit{if}(\neg(\neg b_1 \vee \neg b_2),t,u) \\
\text{(Time8}') & \approx \mathit{if}(\neg b_1 \vee \neg b_2,u,t) \\
\text{(Time9)} & \approx \mathit{if}(\neg b_1,u,\mathit{if}(\neg b_2,u,t)) \\
\text{(Time8}') & \approx \mathit{if}(b_1,\mathit{if}(\neg b_2,u,t),u) \\
\text{(Time8}') & \approx \mathit{if}(b_1,\mathit{if}(b_2,t,u),u)
\end{aligned}
$$

17. $if(b_1, t, if(b_2, u, w)) \approx if(b_1, t, if(\neg b_1 \wedge b_2, u, w))$;

$$
\begin{aligned}
& if(b_1, t, if(b_2, u, w)) \\
& \quad (\text{Time8}') \approx if(\neg b_1, if(b_2, u, w), t) \\
& \quad (\text{Time10}) \approx if(\neg b_1 \wedge b_2, u, if(\neg b_1, w, t)) \\
& \quad (\text{Time8}') \approx if(\neg(\neg b_1 \wedge b_2), if(\neg b_1, w, t), u) \\
& \quad (\text{Time8}') \approx if(\neg(\neg b_1 \wedge b_2), if(b_1, t, w), u) \\
& \quad (\text{Time10}) \approx if(\neg(\neg b_1 \wedge b_2) \wedge b_1, t, if(\neg(\neg b_1 \wedge b_2), w, u)) \\
& \quad (\text{Time8}') \approx if((b_1 \vee \neg b_2) \wedge b_1, t, if(\neg b_1 \wedge b_2, u, w)) \\
& \quad \approx if(b_1, t, if(\neg b_1 \wedge b_2, u, w))
\end{aligned}
$$

18. $if(b_1, if(b_2, u, w), t) \approx if(b_1, if(b_1 \wedge b_2, u, w), t)$;

$$
\begin{aligned}
& if(b_1, if(b_2, u, w), t)(\text{Time8}') \approx if(\neg b_1, t, if(b_2, u, w)) \\
& \quad (17) \approx if(\neg b_1, t, if(b_1 \wedge b_2, u, w)) \\
& \quad (\text{Time8}') \approx if(b_1, if(b_1 \wedge b_2, u, w), t)
\end{aligned}
$$

19. $if(b, t, u) \approx if(b \vee eq(t, u), t, u)$;

$$
\begin{aligned}
& if(b \vee eq(t, u), t, u) \\
& \quad (\text{Time9}) \approx if(b, t, if(eq(t, u), t, u)) \\
& \quad (\text{Time5}) \approx if(b, t, if(t \leq u \wedge u \leq t, t, u)) \\
& \quad (\text{Time11}) \approx if(b, t, u)
\end{aligned}
$$

20. $if(t \leq u, t, u) \approx if(u \leq t, u, t)$;

$$
\begin{aligned}
& if(t \leq u, t, u) \\
& \quad (\text{Time8}') \approx if(\neg\ t \leq u, u, t) \\
& \quad (7) \approx if(\neg\ t \leq u \wedge u \leq t, u, t) \\
& \quad (16) \approx if(u \leq t, if(\neg\ t \leq u, u, t), t) \\
& \quad (\text{Time8}') \approx if(u \leq t, if(t \leq u, t, u), t) \\
& \quad (18) \approx if(u \leq t, if(u \leq t \wedge t \leq u, t, u), t) \\
& \quad (\text{Time11}) \approx if(u \leq t, u, t)
\end{aligned}
$$

21. $min(\mathbf{0}, t) \approx \mathbf{0}$;

$$
\begin{aligned}
& min(\mathbf{0}, t) \\
& (\text{Time6}) \approx if(\mathbf{0} \leq t, \mathbf{0}, t) \\
& (\text{Time2}) \approx if(\mathbf{t}, \mathbf{0}, t) \\
& (\text{Time7}) \approx \mathbf{0}
\end{aligned}
$$

22. $max(\mathbf{0}, t) \approx t$;

$$
\begin{aligned}
& max(\mathbf{0}, t) \\
& (\text{Time6}') \approx if(t \leq \mathbf{0}, \mathbf{0}, t) \\
& \quad (20) \approx if(\mathbf{0} \leq t, t, \mathbf{0}) \\
& (\text{Time2}) \approx if(\mathbf{t}, t, \mathbf{0}) \\
& (\text{Time7}) \approx t
\end{aligned}
$$

23. $min(t,t) \approx t$;

$$\begin{aligned} & min(t,t) \\ (\text{Time6}) &\approx if(t \leq t, t, t) \\ (14) &\approx t \end{aligned}$$

24. $max(t,t) \approx t$;

$$\begin{aligned} & max(t,t) \\ (\text{Time6}') &\approx if(t \leq t, t, t) \\ (14) &\approx t \end{aligned}$$

25. $min(t,u) \approx min(u,t)$;

$$\begin{aligned} & min(t,u) \\ (\text{Time6}) &\approx if(t \leq u, t, u) \\ (20) &\approx if(u \leq t, u, t) \\ (\text{Time6}) &\approx min(u,t) \end{aligned}$$

26. $max(t,u) \approx max(u,t)$;

$$\begin{aligned} & max(t,u) \\ (\text{Time6}') &\approx if(u \leq t, t, u) \\ (20) &\approx if(t \leq u, u, t) \\ (\text{Time6}') &\approx max(u,t) \end{aligned}$$

27. $t \leq min(u,w) \approx t \leq u \wedge t \leq w$;

$$\begin{aligned} & t \leq min(u,w) \\ (\text{Time6}) &\approx t \leq if(u \leq w, u, w) \\ (\text{Time12}) &\approx (u \leq w \wedge t \leq u) \vee (\neg\, u \leq w \wedge t \leq w) \\ (7) &\approx (u \leq w \wedge t \leq u) \vee (\neg\, u \leq w \wedge w \leq u \wedge t \leq w) \\ \text{twice } (\text{Time1}') &\approx (u \leq w \wedge t \leq u \wedge t \leq w) \vee (\neg\, u \leq w \wedge w \leq u \wedge t \leq w \wedge t \leq u) \\ (7) &\approx (u \leq w \wedge t \leq u \wedge t \leq w) \vee (\neg\, u \leq w \wedge t \leq w \wedge t \leq u) \\ &\approx t \leq w \wedge t \leq u \wedge (u \leq w \vee \neg\, u \leq w) \\ &\approx t \leq w \wedge t \leq u \wedge \mathbf{t} \\ &\approx t \leq w \wedge t \leq u \end{aligned}$$

28. $t \leq max(u,w) \approx t \leq u \vee t \leq w$;

$$\begin{aligned} & t \leq max(u,w) \\ (\text{Time6}') &\approx t \leq if(w \leq u, u, w) \\ (\text{Time12}) &\approx (w \leq u \wedge t \leq u) \vee (\neg\, w \leq u \wedge t \leq w) \\ &\approx \mathbf{t} \wedge (w \leq u \vee t \leq w) \wedge (t \leq u \vee \neg\, w \leq u) \wedge (t \leq u \vee t \leq w) \\ (9),(10) &\approx (w \leq u \vee t \leq w \vee t \leq u) \wedge (t \leq u \vee \neg\, w \leq u \vee t \leq w) \wedge (t \leq u \vee t \leq w) \\ &\approx t \leq u \vee t \leq w \end{aligned}$$

29. $min(u,w) \leq t \approx u \leq t \vee w \leq t$;

$$\begin{aligned} & min(u,w) \leq t \\ (\text{Time6}) &\approx if(u \leq w, u, w) \leq t \\ (\text{Time12}) &\approx (u \leq w \wedge u \leq t) \vee (\neg\, u \leq w \wedge w \leq t) \\ &\approx \mathbf{t} \wedge (u \leq w \vee w \leq t) \wedge (u \leq t \vee \neg\, u \leq w) \wedge (u \leq t \vee w \leq t) \\ (9),(11) &\approx (u \leq w \vee w \leq t \vee u \leq t) \wedge (u \leq t \vee \neg\, u \leq w \vee w \leq t) \wedge (u \leq t \vee w \leq t) \\ &\approx u \leq t \vee w \leq t \end{aligned}$$

30. $max(u,w) \leq t \approx u \leq t \wedge w \leq t$;

$$
\begin{aligned}
& max(u,w) \leq t \\
& \qquad (\text{Time6}') \approx if(w \leq u, u, w) \leq t \\
& \qquad (\text{Time13}) \approx (w \leq u \wedge u \leq t) \vee (\neg\, w \leq u \wedge w \leq t) \\
& \qquad (7) \approx (w \leq u \wedge u \leq t) \vee (\neg\, w \leq u \wedge u \leq w \wedge w \leq t) \\
& \text{twice } (\text{Time1}') \approx (w \leq u \wedge u \leq t \wedge w \leq t) \vee (\neg\, w \leq u \wedge u \leq w \wedge w \leq t \wedge u \leq t) \\
& \qquad (7) \approx (w \leq u \wedge u \leq t \wedge w \leq t) \vee (\neg\, w \leq u \wedge w \leq t \wedge u \leq t) \\
& \qquad \approx w \leq t \wedge u \leq t \wedge (w \leq u \vee \neg\, w \leq u) \\
& \qquad \approx w \leq t \wedge u \leq t \wedge \mathbf{t} \\
& \qquad \approx w \leq t \wedge u \leq t
\end{aligned}
$$

31. $t \leq min(t,u) \approx t \leq u$;

$$
\begin{aligned}
& t \leq min(t,u) \\
& \qquad (27) \approx t \leq t \wedge t \leq u \\
& \qquad (2) \approx \mathbf{t} \wedge t \leq u \\
& \qquad \approx t \leq u
\end{aligned}
$$

32. $t \leq max(t,u) \approx \mathbf{t}$;

$$
\begin{aligned}
& t \leq max(t,u) \\
& \qquad (28) \approx t \leq t \vee t \leq u \\
& \qquad (2) \approx \mathbf{t} \vee t \leq u \\
& \qquad \approx \mathbf{t}
\end{aligned}
$$

33. $min(t,u) \leq t \approx \mathbf{t}$;

$$
\begin{aligned}
& min(t,u) \leq t \\
& \qquad (29) \approx t \leq t \vee u \leq t \\
& \qquad (2) \approx \mathbf{t} \vee u \leq t \\
& \qquad \approx \mathbf{t}
\end{aligned}
$$

34. $max(t,u) \leq t \approx u \leq t$;

$$
\begin{aligned}
& max(t,u) \leq t \\
& \qquad (30) \approx t \leq t \wedge u \leq t \\
& \qquad (2) \approx \mathbf{t} \wedge u \leq t \\
& \qquad \approx u \leq t
\end{aligned}
$$

35. $t \leq u \approx eq(t, min(t,u))$;

$$
\begin{aligned}
& eq(t, min(t,u)) \\
& \qquad (\text{Time5}) \approx t \leq min(t,u) \wedge min(t,u) \leq t \\
& \qquad (31),(33) \approx t \leq u \wedge \mathbf{t} \\
& \qquad \approx t \leq u
\end{aligned}
$$

36. $t \leq u \approx eq(u, max(t,u))$;

$$
\begin{aligned}
& eq(u, max(t,u)) \\
& \qquad (26) \approx eq(u, max(u,t)) \\
& \qquad (\text{Time5}) \approx u \leq max(u,t) \wedge max(u,t) \leq u \\
& \qquad (32),(34) \approx \mathbf{t} \wedge t \leq u \\
& \qquad \approx t \leq u
\end{aligned}
$$

37. $min(t, max(t, u)) \approx t$;

$$\begin{aligned} & min(t, max(t, u)) \\ & \qquad (\text{Time6}) \approx if(t \leq max(t, u), t, max(t, u)) \\ & \qquad (32) \approx if(\mathbf{t}, t, max(t, u)) \\ & \qquad (\text{Time7}) \approx t \end{aligned}$$

38. $max(t, min(t, u)) \approx t$;

$$\begin{aligned} & max(t, min(t, u)) \\ & \qquad (\text{Time6}') \approx if(min(t, u) \leq t, t, min(t, u)) \\ & \qquad (33) \approx if(\mathbf{t}, t, min(t, u)) \\ & \qquad (\text{Time7}) \approx t \end{aligned}$$

39. $min(t, min(u, w)) \approx min(min(t, u), w)$;

$$\begin{aligned} & min(t, min(u, w)) \\ & \qquad (\text{Time6}) \approx if(t \leq min(u, w), t, min(u, w)) \\ & \qquad (27), (\text{Time6}) \approx if(t \leq u \wedge t \leq w, t, if(u \leq w, u, w)) \\ & \qquad (17) \approx if(t \leq u \wedge t \leq w, t, if(\neg(t \leq u \wedge t \leq w) \wedge u \leq w, u, w)) \\ & \qquad \approx if(t \leq u \wedge t \leq w, t, if(\neg\, t \leq u \wedge u \leq w, u, w)) \end{aligned}$$

because

$$\begin{aligned} & \neg(t \leq u \wedge t \leq w) \wedge u \leq w \\ & \qquad \approx (\neg\, t \leq u \vee \neg\, t \leq w) \wedge u \leq w \\ & \qquad \approx (\neg\, t \leq u \wedge u \leq w) \vee (\neg\, t \leq w \wedge u \leq w) \\ & \qquad (\text{Time1}'') \approx (\neg\, t \leq u \wedge u \leq w) \vee (\neg\, t \leq w \wedge u \leq w \wedge \neg\, t \leq u) \\ & \qquad \approx \neg\, t \leq u \wedge u \leq w \end{aligned}$$

and

$$\begin{aligned} & min(min(t, u), w) \\ & \qquad (\text{Time6}) \approx if(min(t, u) \leq w, min(t, u), w) \\ & \qquad (29), (\text{Time6}) \approx if(t \leq w \vee u \leq w, if(t \leq u, t, u), w) \\ & \qquad (\text{Time10}) \approx if((t \leq w \vee u \leq w) \wedge t \leq u, t, if(t \leq w \vee u \leq w, u, w)) \\ & \qquad (17) \approx if((t \leq w \vee u \leq w) \wedge t \leq u, t, \\ & \qquad\qquad if(\neg((t \leq w \vee u \leq w) \wedge t \leq u) \wedge (t \leq w \vee u \leq w), u, w)) \\ & \qquad \approx if(t \leq u \wedge t \leq w, t, if(\neg\, t \leq u \wedge u \leq w, u, w)) \end{aligned}$$

because

$$\begin{aligned} & (t \leq w \vee u \leq w) \wedge t \leq u \\ & \qquad \approx (t \leq w \wedge t \leq u) \vee (u \leq w \wedge t \leq u) \\ & \qquad (\text{Time1}') \approx (t \leq w \wedge t \leq u) \vee (u \leq w \wedge t \leq u \wedge t \leq w) \\ & \qquad \approx t \leq w \wedge t \leq u \end{aligned}$$

and

$$\begin{aligned} & \neg((t \leq w \vee u \leq w) \wedge t \leq u) \wedge (t \leq w \vee u \leq w) \\ & \qquad \approx (\neg(t \leq w \vee u \leq w) \vee \neg\, t \leq u) \wedge (t \leq w \vee u \leq w) \\ & \qquad \approx \neg\, t \leq u \wedge (t \leq w \vee u \leq w) \\ & \qquad \approx (\neg\, t \leq u \wedge t \leq w) \vee (\neg\, t \leq u \wedge u \leq w) \\ & \qquad (7), (\text{Time1}') \approx (\neg\, t \leq u \wedge u \leq t \wedge t \leq w \wedge u \leq w) \vee (\neg\, t \leq u \wedge u \leq w) \\ & \qquad \approx \neg\, t \leq u \wedge u \leq w \end{aligned}$$

40. $max(t, max(u,w)) \approx max(max(t,u), w)$. Similar to (39).

41. $min(t, max(u,w)) \approx max(min(t,u), min(t,w))$;

$$
\begin{aligned}
&min(t, max(u,w)) \\
&\qquad (\text{Time6}) \approx if(t \leq max(u,w), t, max(u,w)) \\
&\qquad (28), (\text{Time6}') \approx if(t \leq u \vee t \leq w, t, if(w \leq u, u, w)) \\
&\qquad (17) \approx if(t \leq u \vee t \leq w, t, if(\neg(t \leq u \vee t \leq w) \wedge w \leq u, u, w)) \\
&\qquad \approx if(t \leq u \vee t \leq w, t, if(\neg\, t \leq u \wedge \neg\, t \leq w \wedge w \leq u, u, w))
\end{aligned}
$$

and

$$
\begin{aligned}
&max(min(t,u), min(t,w)) \\
&\qquad (\text{Time6}') \approx if(min(t,w) \leq min(t,u), min(t,u), min(t,w)) \\
&\qquad (27) \approx if(min(t,w) \leq t \wedge min(t,w) \leq u, min(t,u), min(t,w)) \\
&\qquad (33), (29) \approx if(\mathbf{t} \wedge (t \leq u \vee w \leq u), min(t,u), min(t,w)) \\
&\qquad (\text{Time6}) \approx if(t \leq u \vee w \leq u, if(t \leq u, t, u), min(t,w)) \\
&\qquad (\text{Time10}) \approx if((t \leq u \vee w \leq u) \wedge t \leq u, t, if(t \leq u \vee w \leq u, u, min(t,w))) \\
&\qquad (\text{Time8}'), (\text{Time6}) \approx if(t \leq u, t, if(\neg(t \leq u \vee w \leq u), if(t \leq w, t, w), u)) \\
&\qquad (\text{Time10}) \approx if(t \leq u, t, if(\neg(t \leq u \vee w \leq u) \wedge t \leq w, t, if(\neg(t \leq u \vee w \leq u), w, u))) \\
&\qquad (\text{Time8}') \approx if(t \leq u, t, if(\neg\, t \leq u \wedge \neg\, w \leq u \wedge t \leq w, t, if(t \leq u \vee w \leq u, u, w))) \\
&\qquad (\text{Time1}''') \approx if(t \leq u, t, if(\neg\, t \leq u \wedge t \leq w, t, if(t \leq u \vee w \leq u, u, w))) \\
&\qquad (\text{Time9}) \approx if(t \leq u \vee (\neg\, t \leq u \wedge t \leq w), t, if(t \leq u \vee w \leq u, u, w))) \\
&\qquad \approx if(t \leq u \vee t \leq w, t, if(t \leq u \vee w \leq u, u, w))) \\
&\qquad (17) \approx if(t \leq u \vee t \leq w, t, if(\neg(t \leq u \vee t \leq w) \wedge (t \leq u \vee w \leq u), u, w))) \\
&\qquad \approx if(t \leq u \vee t \leq w, t, if(\neg\, t \leq u \wedge \neg\, t \leq w \wedge (t \leq u \vee w \leq u), u, w))) \\
&\qquad \approx if(t \leq u \vee t \leq w, t, if(\neg\, t \leq u \wedge \neg\, t \leq w \wedge w \leq u, u, w))
\end{aligned}
$$

42. $max(t, min(u,w)) \approx min(max(t,u), max(t,w))$. Similar to (41). □

B.2.2 Derivable Identities of Sort *Proc*

Lemma B.2.2. *The following identities are derived with the help of Lemma 2.2.3.*

1. $\sum_{u:Time} x \lhd eq(u,t) \rhd \delta^{\text{\textsc{c}}}\mathbf{0} \approx x$;
2. $\sum_{u:Time} x \lhd u \leq t \rhd \delta^{\text{\textsc{c}}}\mathbf{0} \approx x$;
3. $\sum_{u:Time} x \lhd t \leq u \rhd \delta^{\text{\textsc{c}}}\mathbf{0} \approx x$;
4. $\sum_{u:Time} x \lhd t \leq u \wedge u \leq w \rhd \delta^{\text{\textsc{c}}}\mathbf{0} \approx x \lhd t \leq w \rhd \delta^{\text{\textsc{c}}}\mathbf{0}$;

Proof. The identities are direct applications (by taking $u_0 = t$) of Lemma 2.2.3 adapted to the case of timed μCRL (we have $\delta^{\text{\textsc{c}}}\mathbf{0}$ instead of δ). □

Lemma B.2.3. *The following identities are derivable from the axioms of timed μCRL, booleans and time identities:*

1. $x \lhd c \rhd x \approx x$;

2. $x + x \cdot t \approx x$;
3. $x + \delta \cdot \mathbf{0} \approx x$;
4. $\partial\mathcal{U}(b) \approx \delta$;
5. $\partial\mathcal{U}(x + y) \approx \partial\mathcal{U}(x) + \partial\mathcal{U}(y)$;
6. $\partial\mathcal{U}(x \cdot y) \approx \partial\mathcal{U}(x)$;
7. $\partial\mathcal{U}(x) \cdot y \approx \partial\mathcal{U}(x)$;
8. $\partial\mathcal{U}(\sum_{d:D} p) \approx \sum_{d:D} \partial\mathcal{U}(p)$;
9. $\partial\mathcal{U}(x \lhd c \rhd \delta \cdot \mathbf{0}) \approx \partial\mathcal{U}(x) \lhd c \rhd \delta \cdot \mathbf{0}$;
10. $\partial\mathcal{U}(\partial\mathcal{U}(x)) \approx \partial\mathcal{U}(x)$;
11. $\partial\mathcal{U}(\partial_H(x)) \approx \partial\mathcal{U}(x)$;
12. $\partial\mathcal{U}(\tau_I(x)) \approx \partial\mathcal{U}(x)$;
13. $\partial\mathcal{U}(\rho_R(x)) \approx \partial\mathcal{U}(x)$;
14. $\partial\mathcal{U}(x \ll y) \approx \partial\mathcal{U}(x) \ll y$;
15. $x \ll y \approx x \ll \partial\mathcal{U}(y)$;
16. $b + \delta \approx b$;
17. $\partial\mathcal{U}(x) \cdot t \cdot y \approx \partial\mathcal{U}(x) \cdot t$;
18. $\sum_{u:Time} x \cdot u \lhd eq(u,t) \rhd \delta \cdot \mathbf{0} \approx x \cdot t$;
19. $x \cdot t \cdot u \approx x \cdot u \cdot t$;
20. $x \cdot if(b,t,u) \approx x \cdot t \lhd b \rhd x \cdot u$;
21. $x \cdot min(t,u) \approx x \cdot t \lhd t \leq u \rhd \delta \cdot \mathbf{0} + x \cdot u \lhd u \leq t \rhd \delta \cdot \mathbf{0}$;
22. $(\partial\mathcal{U}(x) \cdot t) \cdot u \approx \partial\mathcal{U}(x) \cdot min(t,u)$;
23. $\sum_{u:Time} \partial\mathcal{U}(x) \cdot u \lhd u \leq t \rhd \delta \cdot \mathbf{0} \approx \partial\mathcal{U}(x) \cdot t$;
24. $\partial\mathcal{U}(x \cdot t) \approx \partial\mathcal{U}(x) \cdot t$;
25. $\partial\mathcal{U}(x) \cdot t + \partial\mathcal{U}(x) \cdot u \approx \partial\mathcal{U}(x) \cdot max(t,u)$;
26. $\sum_{u:Time} \partial\mathcal{U}(x) \cdot u \lhd t \leq u \rhd \delta \cdot \mathbf{0} \approx \partial\mathcal{U}(x)$;
27. $\sum_{u:Time} \partial\mathcal{U}(x) \cdot u \lhd t \leq u \wedge u \leq w \rhd \delta \cdot \mathbf{0} \approx \partial\mathcal{U}(x) \cdot w \lhd t \leq w \rhd \delta \cdot \mathbf{0}$;
28. $x \parallel y \approx y \parallel x$;
29. $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$;

30. $\delta \mathbin{\|\!\_} x \approx \partial\mathcal{U}(x)$;

31. $x \parallel \delta \approx x \cdot \delta$;

32. $x \parallel (y \cdot \delta) \approx (x \parallel y) \cdot \delta$;

33. *if* $\gamma(\mathsf{a},\mathsf{a}')$ *is defined, then* $(\mathsf{a}(\vec{d}) \mid \mathsf{a}'(\vec{d'})) ^{\varsigma}\, t ^{\varsigma}\, t' \approx \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \lhd t = t' \wedge \vec{d} = \vec{d'} \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma}\, min(t,t')$;

34. *if* $\gamma(\mathsf{a},\mathsf{a}')$ *is undefined, then* $(\mathsf{a}(\vec{d}) \mid \mathsf{a}'(\vec{d'})) ^{\varsigma}\, t ^{\varsigma}\, t' \approx \delta ^{\varsigma}\, min(t,t')$;

35. $(\delta \mid b) ^{\varsigma}\, t ^{\varsigma}\, t' \approx \delta ^{\varsigma}\, min(t,t')$;

Proof. 1. $x \lhd c \rhd x \approx x$; Similar to the proof in Lemma 2.2.6.1

2. $x + x ^{\varsigma}\, t \approx x$;

$$\begin{aligned} & x \\ (\text{AT1}) \approx & \sum_{w:\,Time} x ^{\varsigma}\, w \\ (\text{SUM3}) \approx & \sum_{w:\,Time} x ^{\varsigma}\, w + x ^{\varsigma}\, t \\ (\text{AT1}) \approx & \; x + x ^{\varsigma}\, t \end{aligned}$$

3. $x + \delta^{\varsigma}\mathbf{0} \approx x$;

$$\begin{aligned} & x \\ (2) \approx & \; x + x ^{\varsigma}\, \mathbf{0} \\ (\text{A6T}) \approx & \; x + (x + \partial\mathcal{U}(x)) ^{\varsigma}\, \mathbf{0} \\ (\text{ATA2}) \approx & \; x + x ^{\varsigma}\, \mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma}\, \mathbf{0} \\ (2), (\text{ATCC0}) \approx & \; x + (\delta \ll x) ^{\varsigma}\, \mathbf{0} \\ (\text{ATCC7}) \approx & \; x + \delta^{\varsigma}\mathbf{0} \ll x \\ (\text{ATCC1}) \approx & \; x + \delta^{\varsigma}\mathbf{0} \end{aligned}$$

4. $\partial\mathcal{U}(b) \approx \delta$;

$$\begin{aligned} & \partial\mathcal{U}(b) \\ (\text{ATCC0}) \approx & \; \delta \ll b \\ (\text{ATC1}') \approx & \; \delta \end{aligned}$$

5. $\partial\mathcal{U}(x + y) \approx \partial\mathcal{U}(x) + \partial\mathcal{U}(y)$;

$$\begin{aligned} & \partial\mathcal{U}(x+y) \\ (\text{ATCC0}) \approx & \; \delta \ll (x+y) \\ (\text{ATC2}) \approx & \; \delta \ll x + \delta \ll y \\ (\text{ATCC0}) \approx & \; \partial\mathcal{U}(x) + \partial\mathcal{U}(y) \end{aligned}$$

6. $\partial\mathcal{U}(x \cdot y) \approx \partial\mathcal{U}(x)$;

$$\begin{aligned} & \partial\mathcal{U}(x \cdot y) \\ (\text{ATCC0}) \approx & \; \delta \ll (x \cdot y) \\ (\text{ATC3}) \approx & \; \delta \ll x \\ (\text{ATCC0}) \approx & \; \partial\mathcal{U}(x) \end{aligned}$$

7. $\partial\mathcal{U}(x) \cdot y \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
&\partial\mathcal{U}(x) \cdot y \\
(\text{ATCC0}) &\approx (\delta \ll x) \cdot y \\
(\text{ATCC3}) &\approx (\delta \cdot y) \ll x \\
(\text{A7}) &\approx \delta \ll x \\
(\text{ATCC0}) &\approx \partial\mathcal{U}(x)
\end{aligned}$$

8. $\partial\mathcal{U}(\sum_{d:D} p) \approx \sum_{d:D} \partial\mathcal{U}(p)$;

$$\begin{aligned}
&\partial\mathcal{U}(\sum_{d:D} p) \\
(\text{ATCC0}) &\approx \delta \ll (\sum_{d:D} p) \\
(\text{ATC4}) &\approx \sum_{d:D} \delta \ll p \\
(\text{ATCC0}) &\approx \sum_{d:D} \partial\mathcal{U}(p)
\end{aligned}$$

9. $\partial\mathcal{U}(x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) \approx \partial\mathcal{U}(x) \lhd c \rhd \delta^{\varsigma}\mathbf{0}$;

$$\begin{aligned}
&\partial\mathcal{U}(x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) \\
(\text{ATCC0}) &\approx \delta \ll (x \lhd c \rhd \delta^{\varsigma}\mathbf{0}) \\
(\text{ATC5}') &\approx (\delta \ll x) \lhd c \rhd \delta^{\varsigma}\mathbf{0} + \delta^{\varsigma}\mathbf{0} \\
(\text{ATCC0}), (3) &\approx \partial\mathcal{U}(x) \lhd c \rhd \delta^{\varsigma}\mathbf{0}
\end{aligned}$$

10. $\partial\mathcal{U}(\partial\mathcal{U}(x)) \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
&\partial\mathcal{U}(\partial\mathcal{U}(x)) \\
\text{twice } (\text{ATCC0}) &\approx \delta \ll (\delta \ll x) \\
(\text{ATC12}) &\approx (\delta \ll \delta) \ll x \\
(\text{ATC1}') &\approx \delta \ll x \\
(\text{ATCC0}) &\approx \partial\mathcal{U}(x)
\end{aligned}$$

11. $\partial\mathcal{U}(\partial_H(x)) \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
&\partial\mathcal{U}(\partial_H(x)) \\
(\text{ATCC0}) &\approx \delta \ll (\partial_H(x)) \\
(\text{ATC8}) &\approx \delta \ll x \\
(\text{ATCC0}) &\approx \partial\mathcal{U}(x)
\end{aligned}$$

12. $\partial\mathcal{U}(\tau_I(x)) \approx \partial\mathcal{U}(x)$; Similar to (11)

13. $\partial\mathcal{U}(\rho_R(x)) \approx \partial\mathcal{U}(x)$; Similar to (11)

14. $\partial\mathcal{U}(x \ll y) \approx \partial\mathcal{U}(x) \ll y$;

$$\begin{aligned}
&\partial\mathcal{U}(x \ll y) \\
(\text{ATCC0}) &\approx \delta \ll (x \ll y) \\
(\text{ATC12}) &\approx (\delta \ll x) \ll y \\
(\text{ATCC0}) &\approx \partial\mathcal{U}(x) \ll y
\end{aligned}$$

15. $x \ll y \approx x \ll \partial\mathcal{U}(y)$;

$$
\begin{aligned}
& x \ll \partial\mathcal{U}(y) \\
\text{(ATCC0)} &\approx x \ll (\delta \ll x) \\
\text{(ATC12)} &\approx (x \ll \delta) \ll y \\
\text{(ATC1}'\text{)} &\approx x \ll y
\end{aligned}
$$

16. $b + \delta \approx b$;

$$
\begin{aligned}
& b \\
\text{(A6T)} &\approx b + \partial\mathcal{U}(b) \\
(4) &\approx b + \delta
\end{aligned}
$$

17. $\partial\mathcal{U}(x) ^{\varsigma} t \cdot y \approx \partial\mathcal{U}(x) ^{\varsigma} t$;

$$
\begin{aligned}
& \partial\mathcal{U}(x) ^{\varsigma} t \cdot y \\
\text{(ATA3)} &\approx (\partial\mathcal{U}(x) \cdot y) ^{\varsigma} t \\
(7) &\approx \partial\mathcal{U}(x) ^{\varsigma} t
\end{aligned}
$$

18. $\sum_{u:Time} x ^{\varsigma} u \lhd eq(u,t) \rhd \delta^{\varsigma}\mathbf{0} \approx x ^{\varsigma} t$;

$$
\begin{aligned}
& \sum_{u:Time} x ^{\varsigma} u \lhd eq(u,t) \rhd \delta^{\varsigma}\mathbf{0} \\
\text{(PET)} &\approx \sum_{u:Time} x ^{\varsigma} t \lhd eq(u,t) \rhd \delta^{\varsigma}\mathbf{0} \\
\text{(Lemma B.2.2.1)} &\approx x ^{\varsigma} t
\end{aligned}
$$

19. $x ^{\varsigma} t ^{\varsigma} u \approx x ^{\varsigma} u ^{\varsigma} t$;

$$
\begin{aligned}
& x ^{\varsigma} t ^{\varsigma} u \\
\text{(ATA1}'\text{)} &\approx x ^{\varsigma} t \lhd eq(t,u) \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} min(t,u) \\
\text{(PET)} &\approx x ^{\varsigma} u \lhd eq(t,u) \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} min(t,u) \\
\text{(Lemma B.2.1.4\&.25)} &\approx x ^{\varsigma} u \lhd eq(u,t) \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} min(u,t) \\
\text{(ATA1}'\text{)} &\approx x ^{\varsigma} u ^{\varsigma} t
\end{aligned}
$$

20. $x ^{\varsigma} if(b,t,u) \approx x ^{\varsigma} t \lhd b \rhd x ^{\varsigma} u$;

$$
\begin{aligned}
& x ^{\varsigma} if(b,t,u) \\
(18) &\approx \sum_{w:Time} x ^{\varsigma} w \lhd eq(if(b,t,u),w) \rhd \delta^{\varsigma}\mathbf{0} \\
\text{(Lemma B.2.1.5)} &\approx \sum_{w:Time} x ^{\varsigma} w \lhd (b \wedge eq(t,w)) \vee (\neg b \wedge eq(u,w)) \rhd \delta^{\varsigma}\mathbf{0} \\
\text{(Cond5T), (SUM4)} &\approx \sum_{w:Time} x ^{\varsigma} w \lhd b \wedge eq(t,w) \rhd \delta^{\varsigma}\mathbf{0} + \sum_{w:Time} x ^{\varsigma} w \lhd \neg b \wedge eq(u,w) \rhd \delta^{\varsigma}\mathbf{0} \\
\text{twice (Cond4T)} &\approx (\sum_{w:Time} x ^{\varsigma} w \lhd eq(t,w) \rhd \delta^{\varsigma}\mathbf{0}) \lhd b \rhd \delta^{\varsigma}\mathbf{0} \\
& \quad + (\sum_{w:Time} x ^{\varsigma} w \lhd eq(u,w) \rhd \delta^{\varsigma}\mathbf{0}) \lhd \neg b \rhd \delta^{\varsigma}\mathbf{0} \\
\text{twice (18)} &\approx x ^{\varsigma} t \lhd b \rhd \delta^{\varsigma}\mathbf{0} + x ^{\varsigma} u \lhd \neg b \rhd \delta^{\varsigma}\mathbf{0} \\
\text{(Cond3T)} &\approx x ^{\varsigma} t \lhd b \rhd x ^{\varsigma} u
\end{aligned}
$$

21. $x\,{}^{\varsigma}\,min(t,u) \approx x\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + x\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0}$;

$$
\begin{aligned}
& x\,{}^{\varsigma}\,min(t,u) \\
\text{(Time6)} &\approx x\,{}^{\varsigma}\,if(t \leq u, t, u) \\
\text{(A3)} &\approx x\,{}^{\varsigma}\,if(t \leq u, t, u) + x\,{}^{\varsigma}\,if(t \leq u, u, t) \\
\text{(Lemma B.2.1.20)} &\approx x\,{}^{\varsigma}\,if(t \leq u, t, u) + x\,{}^{\varsigma}\,if(u \leq t, t, u) \\
\text{twice (20)} &\approx x\,{}^{\varsigma}\,t \lhd t \leq u \rhd x\,{}^{\varsigma}\,u + x\,{}^{\varsigma}\,u \lhd u \leq t \rhd x\,{}^{\varsigma}\,t \\
\text{twice (Cond3T)} &\approx x\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + x\,{}^{\varsigma}\,u \lhd \neg\, t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} \\
& \quad + x\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} + x\,{}^{\varsigma}\,t \lhd \neg\, u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{twice (Cond5T)} &\approx x\,{}^{\varsigma}\,t \lhd t \leq u \vee \neg\, u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} + x\,{}^{\varsigma}\,u \lhd \neg t \leq u \vee u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{twice (Lemma B.2.1.6)} &\approx x\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + x\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned}
$$

22. $(\partial\mathcal{U}(x)\,{}^{\varsigma}\,t)\,{}^{\varsigma}\,u \approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,min(t,u)$;

$$
\begin{aligned}
& (\partial\mathcal{U}(x)\,{}^{\varsigma}\,t)\,{}^{\varsigma}\,u \\
\text{(ATA1}') &\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd eq(t,u) \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(\partial\mathcal{U}(x))\,{}^{\varsigma}\,min(t,u) \\
\text{(Time5), (10)} &\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \wedge u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,min(t,u) \\
\text{(21)} &\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \wedge u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} \\
& \quad + \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Cond5T)} &\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd (t \leq u \wedge u \leq t) \vee t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} \\
& \quad + \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
&\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(21)} &\approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,min(t,u)
\end{aligned}
$$

23. $\sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \approx \partial\mathcal{U}(x)\,{}^{\varsigma}\,t$;

$$
\begin{aligned}
& \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(SUM3)} &\approx \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq t \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Lemma B.2.1.2),(Cond1)} &\approx \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,t
\end{aligned}
$$

and

$$
\begin{aligned}
& \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \\
\text{(AT1)} &\approx \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,t\,{}^{\varsigma}\,u \\
\text{(22)} &\approx \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,min(t,u) \\
\text{(21)} &\approx \sum_{u:Time} (\partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0}) \\
\text{(SUM4)} &\approx \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,t \lhd t \leq u \rhd \delta{}^{\varsigma}\mathbf{0} + \sum_{u:Time} \partial\mathcal{U}(x)\,{}^{\varsigma}\,u \lhd u \leq t \rhd \delta{}^{\varsigma}\mathbf{0}
\end{aligned}
$$

24. $\partial\mathcal{U}(x \,^{\varsigma}\, t) \approx \partial\mathcal{U}(x) \,^{\varsigma}\, t$;

$$\begin{aligned}
&\partial\mathcal{U}(x \,^{\varsigma}\, t)\\
\text{(ATCC0)} &\approx \delta \ll (x \,^{\varsigma}\, t)\\
\text{(ATC11)} &\approx \sum_{u:Time} (\delta \ll x) \,^{\varsigma}\, u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(ATCC0)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
(23) &\approx \partial\mathcal{U}(x) \,^{\varsigma}\, t
\end{aligned}$$

25. $\partial\mathcal{U}(x) \,^{\varsigma}\, t + \partial\mathcal{U}(x) \,^{\varsigma}\, u \approx \partial\mathcal{U}(x) \,^{\varsigma}\, max(t, u)$;

$$\begin{aligned}
&\partial\mathcal{U}(x) \,^{\varsigma}\, t + \partial\mathcal{U}(x) \,^{\varsigma}\, u\\
\text{twice (23)} &\approx \sum_{w:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd w \leq t \rhd \delta^{\varsigma}\mathbf{0} + \sum_{w:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd w \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(SUM4), (Cond5T)} &\approx \sum_{w:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd w \leq t \vee w \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(Lemma B.2.1.28)} &\approx \sum_{w:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd w \leq max(t, u) \rhd \delta^{\varsigma}\mathbf{0}\\
(23) &\approx \partial\mathcal{U}(x) \,^{\varsigma}\, max(t, u)
\end{aligned}$$

26. $\sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
&\sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(SUM3)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) \,^{\varsigma}\, t \lhd t \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(Lemma B.2.1.2),(Cond1)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) \,^{\varsigma}\, t\\
(23) &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(SUM4), (Cond5T)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \vee u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(Time3), (Cond1)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u\\
\text{(AT1)} &\approx \partial\mathcal{U}(x)
\end{aligned}$$

27. $\sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \wedge u \leq w \rhd \delta^{\varsigma}\mathbf{0} \approx \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd t \leq w \rhd \delta^{\varsigma}\mathbf{0}$;

$$\begin{aligned}
&\sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \wedge u \leq w \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(SUM3)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \wedge u \leq w \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad + \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd t \leq w \wedge w \leq w \rhd \delta^{\varsigma}\mathbf{0}\\
\text{(Lemma B.2.1.2)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{\varsigma}\, u \lhd t \leq u \wedge u \leq w \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) \,^{\varsigma}\, w \lhd t \leq w \rhd \delta^{\varsigma}\mathbf{0}
\end{aligned}$$

and

$$\begin{aligned}
&\partial\mathcal{U}(x) \,{}^{c}\, w \lhd t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
(23) &\approx (\sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd u \leq w \rhd \delta\,{}^{c}\mathbf{0}) \lhd t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
(\text{SUM12T}), (\text{Cond4T}) &\approx \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd u \leq w \wedge t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
(\text{Time3}) &\approx \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd (t \leq u \vee u \leq t) \wedge u \leq w \wedge t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
(\text{Cond5T}), (\text{SUM4}) &\approx \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \wedge u \leq w \wedge t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
&\quad + \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd u \leq t \wedge u \leq w \wedge t \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
(\text{Time1}') &\approx \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \wedge u \leq w \rhd \delta\,{}^{c}\mathbf{0} \\
&\quad + \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd u \leq t \wedge u \leq w \wedge t \leq w \rhd \delta\,{}^{c}\mathbf{0}
\end{aligned}$$

28. $x \parallel y \approx y \parallel x$; Similar to the proof in Lemma 2.2.6.6

29. $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$; Similar to the proof in Lemma 2.2.6.7

30. $\delta \mathbin{⫫} x \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
&\delta \mathbin{⫫} x \\
(\text{AT1}), (\text{SUM6}) &\approx \sum_{t:\,Time} \delta \,{}^{c}\, t \mathbin{⫫} x \\
(\text{CM2T}) &\approx \sum_{t:\,Time} (\delta \,{}^{c}\, t \ll x) \cdot x \\
(\text{ATC8}), (\text{ATA3}) &\approx \sum_{t:\,Time} ((\delta \ll x) \cdot x) \,{}^{c}\, t \\
(\text{AT1}) &\approx (\delta \ll x) \cdot x \\
(\text{ATCC0}), (7) &\approx \partial\mathcal{U}(x)
\end{aligned}$$

31. $x \parallel \delta \approx x \cdot \delta$;

$$\begin{aligned}
&x \parallel \delta \\
(\text{CM1}) &\approx x \mathbin{⫫} \delta + \delta \mathbin{⫫} x + \delta \mid x \\
(\text{SCD1}), (30), (\text{SCDT2}) &\approx x \cdot \delta + \partial\mathcal{U}(x) + \partial\mathcal{U}(x) \\
(\text{A3}), (7) &\approx x \cdot \delta + \partial\mathcal{U}(x) \cdot \delta \\
(\text{A4}) &\approx (x + \partial\mathcal{U}(x)) \cdot \delta \\
(\text{A6T}) &\approx x \cdot \delta
\end{aligned}$$

32. $x \parallel (y \cdot \delta) \approx (x \parallel y) \cdot \delta$;

$$\begin{aligned}
&x \parallel (y \cdot \delta) \\
(31) &\approx x \parallel (y \parallel \delta) \\
(29) &\approx (x \parallel y) \parallel \delta \\
(31) &\approx (x \parallel y) \cdot \delta
\end{aligned}$$

33. if $\gamma(\mathsf{a},\mathsf{a}')$ is defined, then $(\mathsf{a}(\vec{d}) \mid \mathsf{a}'(\vec{d'})) \,^{c}\, t \,^{c}\, t' \approx \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \lhd t = t' \wedge \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} + \delta \,^{c}\, min(t,t')$;

$$
\begin{aligned}
& (\mathsf{a}(\vec{d})|\mathsf{a}'(\vec{d'})) \,^{c}\, t \,^{c}\, t' \\
(\text{CF1}) \approx{} & (\gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \lhd \vec{d} = \vec{d'} \rhd \delta) \,^{c}\, t \,^{c}\, t' \\
(\text{Cond3T}) \approx{} & (\gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \lhd \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} + \delta \lhd \vec{d} \neq \vec{d'} \rhd \delta^{c}\mathbf{0}) \,^{c}\, t \,^{c}\, t' \\
(\text{ATA2}), (\text{ATA5}') \approx{} & \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \,^{c}\, t \,^{c}\, t' \lhd \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} + \delta \,^{c}\, t \,^{c}\, t' \lhd \vec{d} \neq \vec{d'} \rhd \delta^{c}\mathbf{0} \\
(\text{ATA1}'), (4), (22) \approx{} & (\gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \,^{c}\, t \lhd t = t' \rhd \delta^{c}\mathbf{0} + \delta \,^{c}\, min(t,t')) \lhd \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} \\
& + \delta \,^{c}\, min(t,t') \lhd \vec{d} \neq \vec{d'} \rhd \delta^{c}\mathbf{0} \\
(\text{Cond7T}), (\text{Cond4T}) \approx{} & \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \,^{c}\, t \lhd t = t' \wedge \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} \\
& + \delta \,^{c}\, min(t,t') \lhd \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} \\
& + \delta \,^{c}\, min(t,t') \lhd \vec{d} \neq \vec{d'} \rhd \delta^{c}\mathbf{0} \\
(\text{Cond5T}), (\text{Cond1}) \approx{} & \gamma(\mathsf{a},\mathsf{a}')(\vec{d}) \,^{c}\, t \lhd t = t' \wedge \vec{d} = \vec{d'} \rhd \delta^{c}\mathbf{0} + \delta \,^{c}\, min(t,t')
\end{aligned}
$$

34. if $\gamma(\mathsf{a},\mathsf{a}')$ is undefined, then $(\mathsf{a}(\vec{d}) \mid \mathsf{a}'(\vec{d'})) \,^{c}\, t \,^{c}\, t' \approx \delta \,^{c}\, min(t,t')$;

$$
\begin{aligned}
& (\mathsf{a}(\vec{d}) \mid \mathsf{a}'(\vec{d'})) \,^{c}\, t \,^{c}\, t' \\
(\text{CF2}) \approx{} & \delta \,^{c}\, t \,^{c}\, t' \\
(22) \approx{} & \delta \,^{c}\, min(t,t')
\end{aligned}
$$

35. $(\delta \mid b) \,^{c}\, t \,^{c}\, t' \approx \delta \,^{c}\, min(t,t')$;

$$
\begin{aligned}
& (\delta \mid b) \,^{c}\, t \,^{c}\, t' \\
(\text{CD1}) \approx{} & \delta \,^{c}\, t \,^{c}\, t' \\
(22) \approx{} & \delta \,^{c}\, min(t,t')
\end{aligned}
$$

□

Lemma B.2.4. *1.* $\mathbf{0} \ggg x \approx x$;

2. $t \ggg (x \,^{c}\, u) \approx x \,^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, u$;

3. $t \ggg (\partial\mathcal{U}(x) \,^{c}\, u) \approx \partial\mathcal{U}(x) \,^{c}\, u$;

4. $t \ggg \partial\mathcal{U}(x) \approx \partial\mathcal{U}(x)$;

5. $\partial\mathcal{U}(t \ggg x) \approx \partial\mathcal{U}(x)$;

6. $t \ggg (x + y) \approx t \ggg x + t \ggg y$;

7. $t \ggg (x \cdot y) \approx (t \ggg x) \cdot y$;

8. $t \ggg (x \,⌊\!⌊\, y) \approx (t \ggg x) \,⌊\!⌊\, y$;

9. $t \ggg (x \,⌊\!⌊\, y) \approx t \ggg (x \,⌊\!⌊\, t \ggg y)$;

10. $t \ggg (x \mid y) \approx (t \ggg x) \mid y$;

11. $t \ggg (x \mid y) \approx x \mid (t \ggg y)$;

12. $t \ggg \sum_{d:D} p \approx \sum_{d:D} t \ggg p$ *if* d *is not free in* t;

13. $t \ggg (x \lhd c \rhd \delta ^{\varsigma} \mathbf{0}) \approx (t \ggg x) \lhd c \rhd \delta ^{\varsigma} \mathbf{0}$;

14. $t \ggg (\partial_H(x)) \approx \partial_H(t \ggg x)$;

15. $t \ggg (\tau_I(x)) \approx \tau_I(t \ggg x)$;

16. $t \ggg (\rho_R(x)) \approx \rho_R(t \ggg x)$;

17. $t \ggg (x ^{\varsigma} u) \approx (t \ggg x) ^{\varsigma} u$;

18. $t \ggg (u \ggg x) \approx max(t, u) \ggg x$;

19. $t \ggg (u \gg x) \approx u \gg (t \ggg x)$;

20. $t \ggg (x \parallel y) \approx (t \ggg x) \parallel (t \ggg y)$;

21. $t \gg (x ^{\varsigma} u) \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta ^{\varsigma} \mathbf{0} + \delta ^{\varsigma} t$;

22. $t \gg (\delta ^{\varsigma} u) \approx \delta ^{\varsigma} max(t, u)$;

23. $t \gg \delta \approx \delta$;

24. $t \gg (x + y) \approx t \gg x + t \gg y$;

25. $t \gg (x \cdot y) \approx (t \gg x) \cdot y$;

26. $t \gg \sum_{d:D} p \approx \sum_{d:D} t \gg p$ *if* d *is not free in* t;

27. $t \gg (x \lhd c \rhd \delta ^{\varsigma} \mathbf{0}) \approx (t \gg x) \lhd c \rhd \delta ^{\varsigma} \mathbf{0} + \delta ^{\varsigma} t$;

28. $t \gg (\partial_H(x)) \approx \partial_H(t \gg x)$;

29. $t \gg (\tau_I(x)) \approx \tau_I(t \gg x)$;

30. $t \gg (\rho_R(x)) \approx \rho_R(t \gg x)$;

31. $t \gg (u \gg x) \approx max(t, u) \gg x$;

32. *if* $x \approx t \gg x$, *then* $x \approx t \ggg x$;

33. *if* $x \approx t \gg x$ *and* $y \approx t \gg y$, *then* $t \gg (x \parallel y) \approx x \parallel y$;

34. $x ^{\varsigma} t \ll \delta ^{\varsigma} u \approx x ^{\varsigma} t \lhd t \leq u \rhd \delta ^{\varsigma} \mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} min(t, u)$;

35. $\partial\mathcal{U}(x) ^{\varsigma} t \ll \delta ^{\varsigma} u \approx \partial\mathcal{U}(x) ^{\varsigma} min(t, u)$;

36. $(\partial\mathcal{U}(x) ^{\varsigma} t \ll y) \cdot z \approx \partial\mathcal{U}(x) ^{\varsigma} t \ll y$;

37. $b ^{\varsigma} t \cdot y \approx b ^{\varsigma} t \cdot t \ggg y$;

38. $b ^{\varsigma} t \mathbin{\|\!\lfloor} y \approx (b ^{\varsigma} t \ll y) \cdot (t \ggg y)$;

39. if $x \approx t \ggg x$, *then* $(b \,{}^{c}\, t \cdot x) \mathbin{|\!\!\lfloor} y \approx (b \,{}^{c}\, t \ll y) \cdot (x \parallel t \ggg y)$;

Proof. 1. $\mathbf{0} \ggg x \approx x$;

$$
\begin{aligned}
& \mathbf{0} \ggg x \\
\text{(ATD0)} \approx & \sum_{t:\,Time} x \,{}^{c}\, t \lhd \mathbf{0} \leq t \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Time2), (Cond1)} \approx & \sum_{t:\,Time} x \,{}^{c}\, t \\
\text{(AT1)} \approx & \; x
\end{aligned}
$$

2. $t \ggg (x \,{}^{c}\, u) \approx x \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} + \partial\mathcal{U}(x) \,{}^{c}\, u$;

$$
\begin{aligned}
& t \ggg (x \,{}^{c}\, u) \\
\text{(ATD0)} \approx & \sum_{w:\,Time} (x \,{}^{c}\, u) \,{}^{c}\, w \lhd t \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(ATA1}'\text{), (Cond4T),} \approx & \sum_{w:\,Time} x \,{}^{c}\, u \lhd u = w \wedge t \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Cond7T), (SUM4)} \quad & + \sum_{w:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, min(u,w) \lhd t \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Cond7T),(Lemma B.2.3.21)} \approx & \sum_{w:\,Time} (x \,{}^{c}\, u \lhd t \leq w \rhd \delta \,{}^{c}\mathbf{0}) \lhd u = w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Cond4T), (SUM4)} \quad & + \sum_{w:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd u \leq w \wedge t \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
& + \sum_{w:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, w \lhd w \leq u \wedge t \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(PET),} \approx & \sum_{w:\,Time} (x \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0}) \lhd u = w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Lemma B.2.1.30),} \quad & + \sum_{w:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd max(u,t) \leq w \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Lemma B.2.3.27)} \quad & + \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Lemma B.2.3.18),} \approx & \; x \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Lemma B.2.2.3)} \quad & + \partial\mathcal{U}(x) \,{}^{c}\, u + \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Cond1), (Cond5T)} \approx & \; x \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} + \partial\mathcal{U}(x) \,{}^{c}\, u
\end{aligned}
$$

3. $t \ggg (\partial\mathcal{U}(x) \,{}^{c}\, u) \approx \partial\mathcal{U}(x) \,{}^{c}\, u$;

$$
\begin{aligned}
& t \ggg (\partial\mathcal{U}(x) \,{}^{c}\, u) \\
\text{(2)} \approx & \; \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} + \partial\mathcal{U}(\partial\mathcal{U}(x)) \,{}^{c}\, u \\
\text{(Lemma B.2.3.10), (Cond1), (Cond5T)} \approx & \; \partial\mathcal{U}(x) \,{}^{c}\, u
\end{aligned}
$$

4. $t \ggg \partial\mathcal{U}(x) \approx \partial\mathcal{U}(x)$;

$$
\begin{aligned}
& t \ggg \partial\mathcal{U}(x) \\
\text{(ATD0)} \approx & \sum_{u:\,Time} \partial\mathcal{U}(x) \,{}^{c}\, u \lhd t \leq u \rhd \delta \,{}^{c}\mathbf{0} \\
\text{(Lemma B.2.3.26)} \approx & \; \partial\mathcal{U}(x)
\end{aligned}
$$

5. $\partial\mathcal{U}(t \ggg x) \approx \partial\mathcal{U}(x)$;

$$\begin{aligned}
& \partial\mathcal{U}(t \ggg x) \\
\text{(ATD0)} &\approx \partial\mathcal{U}(\sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0}) \\
\text{(Lemma B.2.3.8,.9\&.24)} &\approx \sum_{u:Time} \partial\mathcal{U}(x) \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(Lemma B.2.3.26)} &\approx \partial\mathcal{U}(x)
\end{aligned}$$

6. $t \ggg (x+y) \approx t \ggg x + t \ggg y$;

$$\begin{aligned}
& t \ggg (x+y) \\
\text{(ATD0)} &\approx \sum_{u:Time} (x+y) \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(ATA2), (Cond7T), (SUM4)} &\approx \sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} + \sum_{u:Time} y \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{twice (ATD0)} &\approx t \ggg x + t \ggg y
\end{aligned}$$

7. $t \ggg (x \cdot y) \approx (t \ggg x) \cdot y$;

$$\begin{aligned}
& t \ggg (x \cdot y) \\
\text{(ATD0)} &\approx \sum_{u:Time} (x \cdot y) \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(ATA3)} &\approx \sum_{u:Time} x \,^{c}\, u \cdot y \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(Cond6T), (SUM5)} &\approx (\sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0}) \cdot y \\
\text{(ATD0)} &\approx (t \ggg x) \cdot y
\end{aligned}$$

8. $t \ggg (x \mathbin{\|\!\!\_} y) \approx (t \ggg x) \mathbin{\|\!\!\_} y$;

$$\begin{aligned}
& (t \ggg x) \mathbin{\|\!\!_} y \\
\text{(ATD0)} &\approx (\sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0}) \mathbin{\|\!\!_} y \\
\text{(SUM6), (Cond8T), (SCT1)} &\approx \sum_{u:Time} (x \mathbin{\|\!\!_} y) \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(ATD0)} &\approx t \ggg (x \mathbin{\|\!\!_} y)
\end{aligned}$$

9. $t \ggg (x \mathbin{\|\!\!\_} y) \approx t \ggg (x \mathbin{\|\!\!\_} t \ggg y)$;

$$\begin{aligned}
& (t \ggg x) \mathbin{\|\!\!_} y \\
\text{(ATD0)} &\approx (\sum_{u:Time} x \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0}) \mathbin{\|\!\!_} y \\
\text{(SUM6), (Cond8T)} &\approx \sum_{u:Time} (x \,^{c}\, u \mathbin{\|\!\!_} y) \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(SCT2)} &\approx \sum_{u:Time} (x \,^{c}\, u \mathbin{\|\!\!_} t \ggg y) \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(SCT1)} &\approx \sum_{u:Time} (x \mathbin{\|\!\!_} t \ggg y) \,^{c}\, u \lhd t \leq u \rhd \delta\,^{c}\mathbf{0} \\
\text{(ATD0)} &\approx t \ggg (x \mathbin{\|\!\!_} t \ggg y)
\end{aligned}$$

10. $t \ggg (x \mid y) \approx (t \ggg x) \mid y$;

$$\begin{aligned}
& (t \ggg x) \mid y \\
\text{(ATD0)} \approx\ & (\sum_{u:Time} x \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \mid y \\
\text{(SUM7), (Cond9T), (ATA7)} \approx\ & \sum_{u:Time} (x \mid y) \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(ATD0)} \approx\ & t \ggg (x \mid y)
\end{aligned}$$

11. $t \ggg (x \mid y) \approx x \mid (t \ggg y)$;

$$\begin{aligned}
& x \mid (t \ggg y) \\
\text{(ATD0)} \approx\ & x \mid (\sum_{u:Time} y \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \\
\text{(SUM7}'\text{), (Cond9}'\text{), (ATA8)} \approx\ & \sum_{u:Time} (x \mid y) \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(ATD0)} \approx\ & t \ggg (x \mid y)
\end{aligned}$$

12. $t \ggg \sum_{d:D} p \approx \sum_{d:D} t \ggg p$ if d is not free in t;

$$\begin{aligned}
& t \ggg \sum_{d:D} p \\
\text{(ATD0)} \approx\ & \sum_{u:Time} (\sum_{d:D} p) \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(ATA4), (SUM12T)} \approx\ & \sum_{u:Time} \sum_{d:D} p \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Lemma 2.2.5)} \approx\ & \sum_{d:D} (\sum_{u:Time} p \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \\
\text{(ATD0)} \approx\ & \sum_{d:D} (t \ggg p)
\end{aligned}$$

13. $t \ggg (x \lhd c \rhd \delta^{c}\mathbf{0}) \approx (t \ggg x) \lhd c \rhd \delta^{c}\mathbf{0}$;

$$\begin{aligned}
& t \ggg (x \lhd c \rhd \delta^{c}\mathbf{0}) \\
\text{(ATD0)} \approx\ & \sum_{u:Time} (x \lhd c \rhd \delta^{c}\mathbf{0}) \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(ATA5}'\text{), (Cond4T)} \approx\ & \sum_{u:Time} x \,{}^{c}\, u \lhd c \wedge t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Cond4T), (SUM12T)} \approx\ & (\sum_{u:Time} x \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \lhd c \rhd \delta^{c}\mathbf{0} \\
\text{(ATD0)} \approx\ & (t \ggg x) \lhd c \rhd \delta^{c}\mathbf{0}
\end{aligned}$$

14. $t \ggg (\partial_H(x)) \approx \partial_H(t \ggg x)$;

$$\begin{aligned}
& t \ggg (\partial_H(x)) \\
\text{(ATD0)} \approx\ & \sum_{u:Time} \partial_H(x) \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(D7), (D5T), (SUM8)} \approx\ & \partial_H(\sum_{u:Time} x \,{}^{c}\, u \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \\
\text{(ATD0)} \approx\ & \partial_H(t \ggg x)
\end{aligned}$$

15. $t \ggg (\tau_I(x)) \approx \tau_I(t \ggg x)$; Similar to (14)

16. $t \ggg (\rho_R(x)) \approx \rho_R(t \ggg x)$; Similar to (14)

17. $t \ggg (x ^{\varsigma} u) \approx (t \ggg x) ^{\varsigma} u$;

$$
\begin{aligned}
& t \ggg (x ^{\varsigma} u) \\
\text{(ATD0)} \approx & \sum_{w:Time} x ^{\varsigma} u ^{\varsigma} w \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Lemma B.2.3.19)} \approx & \sum_{w:Time} x ^{\varsigma} w ^{\varsigma} u \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(ATA5}'\text{), (ATA4)} \approx & (\sum_{w:Time} x ^{\varsigma} w \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0}) ^{\varsigma} u \\
\text{(ATD0)} \approx & (t \ggg x) ^{\varsigma} u
\end{aligned}
$$

18. $t \ggg (u \ggg x) \approx max(t, u) \ggg x$;

$$
\begin{aligned}
& t \ggg (u \ggg x) \\
\text{twice (ATD0)} \approx & \sum_{w:Time} (\sum_{w':Time} x ^{\varsigma} w' \lhd u \leq w' \rhd \delta ^{\varsigma} \mathbf{0}) ^{\varsigma} w \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(ATA4), (ATA5}'\text{),} \approx & \sum_{w:Time} \sum_{w':Time} x ^{\varsigma} w' ^{\varsigma} w \lhd u \leq w' \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(SUM12T), (Cond4T)} & \\
\text{(ATA1}'\text{), (Lemma B.2.3.21),} \approx & \sum_{w:Time} \sum_{w':Time} x ^{\varsigma} w \lhd w = w' \wedge u \leq w' \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Cond7T), (SUM4),} & + \sum_{w:Time} \sum_{w':Time} \partial\mathcal{U}(x) ^{\varsigma} w \lhd w \leq w' \wedge u \leq w' \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Cond4T)} & + \sum_{w:Time} \sum_{w':Time} \partial\mathcal{U}(x) ^{\varsigma} w' \lhd w' \leq w \wedge u \leq w' \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Cond4T), (SUM12T),} \approx & \sum_{w:Time} \sum_{w':Time} (x ^{\varsigma} w \lhd u \leq w' \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0}) \lhd w = w' \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Lemma B.2.1.30),} & + \sum_{w:Time} (\sum_{w':Time} \partial\mathcal{U}(x) ^{\varsigma} w \lhd max(w, u) \leq w' \rhd \delta ^{\varsigma} \mathbf{0}) \\
& \qquad \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
& + \sum_{w:Time} (\sum_{w':Time} \partial\mathcal{U}(x) ^{\varsigma} w' \lhd u \leq w' \wedge w' \leq w \rhd \delta ^{\varsigma} \mathbf{0}) \\
& \qquad \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(PET)} \approx & \sum_{w:Time} \sum_{w':Time} (x ^{\varsigma} w \lhd u \leq w \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0}) \lhd w = w' \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Lemma B.2.2.3),} & + \sum_{w:Time} \partial\mathcal{U}(x) ^{\varsigma} w \lhd t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Lemma B.2.3.27),(Cond4T)} & + \sum_{w:Time} \partial\mathcal{U}(x) ^{\varsigma} w \lhd u \leq w \wedge t \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(Lemma B.2.3.18\&.26),} \approx & \sum_{w:Time} x ^{\varsigma} w \lhd max(u, t) \leq w \rhd \delta ^{\varsigma} \mathbf{0} + \partial\mathcal{U}(x) \\
\text{(Lemma B.2.1.30)} & + \sum_{w:Time} \partial\mathcal{U}(x) ^{\varsigma} w \lhd max(u, t) \leq w \rhd \delta ^{\varsigma} \mathbf{0} \\
\text{(ATD0), (Lemma B.2.3.26)} \approx & max(u, t) \ggg x + \partial\mathcal{U}(x) + \partial\mathcal{U}(x) \\
\text{(A3)} \approx & max(u, t) \ggg x + \partial\mathcal{U}(x)
\end{aligned}
$$

and

$$
\begin{aligned}
&max(t,u) \ggg x\\
&\quad (\text{A6T}) \approx max(t,u) \ggg (x + \partial\mathcal{U}(x))\\
&\quad (6),(4) \approx max(t,u) \ggg x + \partial\mathcal{U}(x)
\end{aligned}
$$

19. $t \ggg (u \gg x) \approx u \gg (t \ggg x)$;

$$
\begin{aligned}
&t \ggg (u \gg x)\\
&\quad (\text{ATB0}) \approx t \ggg (u \ggg x + \delta ^{\varsigma} u)\\
&\quad (6) \approx t \ggg (u \ggg x) + t \ggg (\delta ^{\varsigma} u)\\
&\quad (18),(3) \approx max(t,u) \ggg x + \delta ^{\varsigma} u\\
&\quad (\text{Lemma B.2.1.26}), \approx max(u,t) \ggg x + \delta ^{\varsigma} u\\
&\quad (18) \approx u \ggg t \ggg x + \delta ^{\varsigma} u\\
&\quad (\text{ATB0}) \approx u \gg t \ggg x
\end{aligned}
$$

20. $t \ggg (x \parallel y) \approx (t \ggg x) \parallel (t \ggg y)$;

$$
\begin{aligned}
&t \ggg (x \parallel y)\\
&\quad (\text{CM1}) \approx t \ggg (x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x + x \mid y)\\
&\quad \text{twice } (6) \approx t \ggg (x \mathbin{\|\!\_} y) + t \ggg (y \mathbin{\|\!\_} x) + t \ggg (x \mid y)\\
&\quad \text{twice } (8),(\text{Lemma B.2.1.24}) \approx t \ggg (x \mathbin{\|\!\_} t \ggg y) + t \ggg (y \mathbin{\|\!\_} t \ggg x) + max(t,t) \ggg (x \mid y)\\
&\quad \text{twice } (8),(18) \approx (t \ggg x) \mathbin{\|\!\_} (t \ggg y) + (t \ggg y) \mathbin{\|\!\_} (t \ggg x) + t \ggg t \ggg (x \mid y)\\
&\quad (10),(11) \approx (t \ggg x) \mathbin{\|\!\_} (t \ggg y) + (t \ggg y) \mathbin{\|\!\_} (t \ggg x) + (t \ggg x) \mid (t \ggg y)\\
&\quad (\text{CM1}) \approx (t \ggg x) \parallel (t \ggg y)
\end{aligned}
$$

21. $t \gg (x ^{\varsigma} u) \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma} t$;

$$
\begin{aligned}
&t \gg (x ^{\varsigma} u)\\
&\quad (\text{ATB0}) \approx t \ggg (x ^{\varsigma} u) + \delta ^{\varsigma} t\\
&\quad (2),(\text{Lemma B.2.3.23}) \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} u + \sum_{u:\,Time} \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad (\text{Cond1}),(\text{Time2}), \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \partial\mathcal{U}(x) ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad (\text{Cond5T}),(\text{SUM3}) \qquad + \partial\mathcal{U}(x) ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad\qquad + \sum_{u:\,Time} \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad (\text{Cond7T}),(\text{ATA2}) \approx (x + \partial\mathcal{U}(x)) ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad\qquad + (\partial\mathcal{U}(x) + \delta) ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad\qquad + \sum_{u:\,Time} \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad (\text{A6T}),(\text{A6T}'), \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad\qquad + \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\qquad\qquad + \sum_{u:\,Time} \delta ^{\varsigma} u \lhd u \leq t \rhd \delta^{\varsigma}\mathbf{0}\\
&\quad (\text{SUM3}),(\text{Lemma B.2.3.23}) \approx x ^{\varsigma} u \lhd t \leq u \rhd \delta^{\varsigma}\mathbf{0} + \delta ^{\varsigma} t
\end{aligned}
$$

22. $t \gg (\delta ^{\varsigma} u) \approx \delta ^{\varsigma} max(t,u)$;

$$
\begin{aligned}
&t \gg (\delta ^{\varsigma} u)\\
&\quad (\text{ATB0}) \approx t \ggg (\delta ^{\varsigma} u) + \delta ^{\varsigma} t\\
&\quad (3) \approx \delta ^{\varsigma} u + \delta ^{\varsigma} t\\
&\quad (\text{Lemma B.2.3.25}) \approx \delta ^{\varsigma} max(t,u)
\end{aligned}
$$

23. $t \gg \delta \approx \delta$;

$$\begin{aligned} & t \gg \delta \\ \text{(ATB0)} & \approx t \ggg \delta + \delta \,^{\varsigma}\, t \\ (4) & \approx \delta + \delta \,^{\varsigma}\, t \\ \text{(Lemma B.2.3.2)} & \approx \delta \end{aligned}$$

24. $t \gg (x + y) \approx t \gg x + t \gg y$;

$$\begin{aligned} & t \gg (x + y) \\ \text{(ATB0)} & \approx t \ggg (x + y) + \delta \,^{\varsigma}\, t \\ (6), \text{(A3)} & \approx t \ggg x + t \ggg y + \delta \,^{\varsigma}\, t + \delta \,^{\varsigma}\, t \\ \text{twice (ATB0)} & \approx t \gg x + t \gg y \end{aligned}$$

25. $t \gg (x \cdot y) \approx (t \gg x) \cdot y$;

$$\begin{aligned} & t \gg (x \cdot y) \\ \text{(ATB0)} & \approx t \ggg (x \cdot y) + \delta \,^{\varsigma}\, t \\ (7),\text{(Lemma B.2.3.17)} & \approx (t \ggg x) \cdot y + \delta \,^{\varsigma}\, t \cdot y \\ \text{(A4)} & \approx (t \ggg x + \delta \,^{\varsigma}\, t) \cdot y \\ \text{(ATB0)} & \approx (t \gg x) \cdot y \end{aligned}$$

26. $t \gg \sum_{d:D} p \approx \sum_{d:D} t \gg p$ if d is not free in t;

$$\begin{aligned} & t \gg \sum_{d:D} p \\ \text{(ATB0)} & \approx t \ggg (\sum_{d:D} p) + \delta \,^{\varsigma}\, t \\ (12), \text{(SUM1)} & \approx \sum_{d:D} t \ggg p + \sum_{d:D} \delta \,^{\varsigma}\, t \\ \text{(SUM4)} & \approx \sum_{d:D} (t \ggg p + \delta \,^{\varsigma}\, t) \\ \text{(ATB0)} & \approx \sum_{d:D} t \gg p \end{aligned}$$

27. $t \gg (x \lhd c \rhd \delta \,^{\varsigma}\mathbf{0}) \approx (t \gg x) \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t$;

$$\begin{aligned} & t \gg (x \lhd c \rhd \delta \,^{\varsigma}\mathbf{0}) \\ \text{(ATB0)} & \approx t \ggg (x \lhd c \rhd \delta \,^{\varsigma}\mathbf{0}) + \delta \,^{\varsigma}\, t \\ (13), \text{(Cond1)} & \approx (t \ggg x) \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \lhd \neg c \rhd \delta \,^{\varsigma}\mathbf{0} \\ \text{(A3)} & \approx (t \ggg x) \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \lhd \neg c \rhd \delta \,^{\varsigma}\mathbf{0} \\ \text{(Cond7T)} & \approx (t \ggg x + \delta \,^{\varsigma}\, t) \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \\ \text{(ATB0)} & \approx (t \gg x) \lhd c \rhd \delta \,^{\varsigma}\mathbf{0} + \delta \,^{\varsigma}\, t \end{aligned}$$

28. $t \gg (\partial_H(x)) \approx \partial_H(t \gg x)$;

$$\begin{aligned} & t \gg (\partial_H(x)) \\ \text{(ATB0)} & \approx t \ggg (\partial_H(x)) + \delta \,^{\varsigma}\, t \\ (14), \text{(D2)}, \text{(D7)} & \approx \partial_H(t \ggg x) + \partial_H(\delta \,^{\varsigma}\, t) \\ \text{(D3)} & \approx \partial_H(t \ggg x + \delta \,^{\varsigma}\, t) \\ \text{(ATB0)} & \approx \partial_H(t \gg x) \end{aligned}$$

29. $t \gg (\tau_I(x)) \approx \tau_I(t \gg x)$; Similar to (28).

30. $t \gg (\rho_R(x)) \approx \rho_R(t \gg x)$; Similar to (28).

31. $t \gg (u \gg x) \approx max(t,u) \gg x$;

$$\begin{aligned}
& t \gg (u \gg x)\\
\text{twice (ATB0)} &\approx t \ggg (u \ggg x + \delta \mathbin{{}^{\varsigma}} u) + \delta \mathbin{{}^{\varsigma}} t\\
(6),(3) &\approx t \ggg (u \ggg x) + \delta \mathbin{{}^{\varsigma}} u + \delta \mathbin{{}^{\varsigma}} t\\
\text{(18),(Lemma B.2.3.25)} &\approx max(t,u) \ggg x + \delta \mathbin{{}^{\varsigma}} max(t,u)\\
\text{(ATB0)} &\approx max(t,u) \gg x
\end{aligned}$$

32. if $x \approx t \gg x$, then $x \approx t \ggg x$;

$$\begin{aligned}
& t \ggg x\\
\text{(assumption)} &\approx t \ggg (t \gg x)\\
\text{(ATB0)} &\approx t \ggg (t \ggg x + \delta \mathbin{{}^{\varsigma}} t)\\
(6) &\approx t \ggg t \ggg x + t \ggg \delta \mathbin{{}^{\varsigma}} t\\
\text{(18),(Lemma B.2.1.24),(3)} &\approx t \ggg x + \delta \mathbin{{}^{\varsigma}} t\\
\text{(ATB0)} &\approx t \gg x\\
\text{(assumption)} &\approx x
\end{aligned}$$

33. if $x \approx t \gg x$ and $y \approx t \gg y$, then $t \gg (x \parallel y) \approx x \parallel y$;

$$\begin{aligned}
& t \gg (x \parallel y)\\
\text{(ATB0)} &\approx t \ggg (x \parallel y) + \delta \mathbin{{}^{\varsigma}} t\\
(18) &\approx (t \ggg x) \parallel (t \ggg y) + \delta \mathbin{{}^{\varsigma}} t\\
\text{twice (32)} &\approx x \parallel y + \delta \mathbin{{}^{\varsigma}} t\\
&\approx x \parallel y
\end{aligned}$$

because

$$\begin{aligned}
& x \parallel y\\
\text{(CM1),assumption} &\approx x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + (t \gg x) \mid (t \gg y)\\
\text{twice (ATD0)} &\approx x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + (t \ggg x + \delta \mathbin{{}^{\varsigma}} t) \mid (t \ggg y + \delta \mathbin{{}^{\varsigma}} t)\\
\text{(CM8), (CM9)} &\approx x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + t \ggg x \mid (t \ggg y + \delta \mathbin{{}^{\varsigma}} t)\\
&\qquad + \delta \mathbin{{}^{\varsigma}} t \mid t \ggg y + \delta \mathbin{{}^{\varsigma}} t \mid \delta \mathbin{{}^{\varsigma}} t\\
\text{(A3),this derivation} &\approx x \parallel y + \delta \mathbin{{}^{\varsigma}} t \mid \delta \mathbin{{}^{\varsigma}} t\\
\text{(ATA7), (ATA8)} &\approx x \parallel y + (\delta \mid \delta) \mathbin{{}^{\varsigma}} t \mathbin{{}^{\varsigma}} t\\
\text{(Lemma B.2.3.35),(Lemma B.2.1.23)} &\approx x \parallel y + \delta \mathbin{{}^{\varsigma}} t
\end{aligned}$$

34. $x \,^{c}\, t \ll \delta \,^{c}\, u \approx x \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, min(t,u)$;

$$\begin{aligned}
& x \,^{c}\, t \ll \delta \,^{c}\, u \\
\text{(ATC12), (ATC1}') \approx{} & \sum_{w:Time} x \,^{c}\, t \,^{c}\, w \lhd w \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(ATA1}') \approx{} & \sum_{w:Time} \left(x \,^{c}\, t \lhd t = w \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, min(t,w)\right) \lhd w \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Lemma B.2.3.21),(Cond7T),} \approx{} & \sum_{w:Time} x \,^{c}\, t \lhd t = w \wedge w \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Cond4T), (SUM4)} \quad & + \sum_{w:Time} \partial\mathcal{U}(x) \,^{c}\, t \lhd t \leq w \wedge w \leq u \rhd \delta^{c}\mathbf{0} \\
& + \sum_{w:Time} \partial\mathcal{U}(x) \,^{c}\, w \lhd w \leq t \wedge w \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Cond7T),} \approx{} & \sum_{w:Time} (x \,^{c}\, t \lhd w \leq u \rhd \delta^{c}\mathbf{0}) \lhd t = w \rhd \delta^{c}\mathbf{0} \\
\text{(Lemma B.2.3.27),} \quad & + \partial\mathcal{U}(x) \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0} \\
\text{(Lemma B.2.1.27)} \quad & + \sum_{w:Time} \partial\mathcal{U}(x) \,^{c}\, w \lhd w \leq min(t,u) \rhd \delta^{c}\mathbf{0} \\
\text{(PET),(Lemma B.2.3.23)} \approx{} & \sum_{w:Time} (x \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0}) \lhd t = w \rhd \delta^{c}\mathbf{0} \\
& + \partial\mathcal{U}(x) \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, min(t,u) \\
\text{(Lemma B.2.3.18\&.21)} \approx{} & x \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(x) \,^{c}\, min(t,u)
\end{aligned}$$

35. $\partial\mathcal{U}(x) \,^{c}\, t \ll \delta \,^{c}\, u \approx \partial\mathcal{U}(x) \,^{c}\, min(t,u)$;

$$\begin{aligned}
& \partial\mathcal{U}(x) \,^{c}\, t \ll \delta \,^{c}\, u \\
\text{(34)} \approx{} & \partial\mathcal{U}(x) \,^{c}\, t \lhd t \leq u \rhd \delta^{c}\mathbf{0} + \partial\mathcal{U}(\partial\mathcal{U}(x)) \,^{c}\, min(t,u) \\
\text{(Lemma B.2.3.10\&.21)} \approx{} & \partial\mathcal{U}(x) \,^{c}\, min(t,u)
\end{aligned}$$

36. $(\partial\mathcal{U}(x) \,^{c}\, t \ll y) \cdot z \approx \partial\mathcal{U}(x) \,^{c}\, t \ll y$;

$$\begin{aligned}
& (\partial\mathcal{U}(x) \,^{c}\, t \ll y) \cdot z \\
\text{(ATCC7)} \approx{} & (\partial\mathcal{U}(x) \,^{c}\, t \cdot z) \ll y \\
\text{(Lemma B.2.3.17)} \approx{} & \partial\mathcal{U}(x) \,^{c}\, t \ll y
\end{aligned}$$

37. $b \,^{c}\, t \cdot y \approx b \,^{c}\, t \cdot t \ggg y$;

$$\begin{aligned}
& b \,^{c}\, t \cdot y \\
\text{(AT2)} \approx{} & b \,^{c}\, t \cdot t \gg y \\
\text{(ATB0)} \approx{} & b \,^{c}\, t \cdot (t \ggg y + \delta \,^{c}\, t) \\
\text{(Lemma B.2.1.24),(18)} \approx{} & b \,^{c}\, t \cdot (t \ggg t \ggg y + \delta \,^{c}\, t) \\
\text{(ATB0)} \approx{} & b \,^{c}\, t \cdot t \gg t \ggg y \\
\text{(AT2)} \approx{} & b \,^{c}\, t \cdot t \ggg y
\end{aligned}$$

38. $b \,^{c}\, t \mathbin{|\!\!\lfloor} y \approx (b \,^{c}\, t \ll y) \cdot (t \ggg y)$;

$$\begin{aligned}
& b \,^{c}\, t \mathbin{|\!\!\lfloor} y \\
\text{(CM2T)} \approx{} & (b \,^{c}\, t \ll y) \cdot y \\
\text{(ATCC7)} \approx{} & (b \,^{c}\, t \cdot y) \ll y \\
\text{(37)} \approx{} & (b \,^{c}\, t \cdot t \ggg y) \ll y \\
\text{(ATCC7)} \approx{} & (b \,^{c}\, t \ll y) \cdot (t \ggg y)
\end{aligned}$$

39. if $x \approx t \gg x$, then $(b \,^c\, t \cdot x) \mathbin{\|\!\_} y \approx (b \,^c\, t \ll y) \cdot (x \parallel t \ggg y)$;

$$
\begin{aligned}
& (b \,^c\, t \cdot x) \mathbin{\|\!\_} y \\
\text{(CM3T)} \approx\ & (b \,^c\, t \ll y) \cdot ((t \gg x) \parallel y) \\
\text{(ATCC7)} \approx\ & (b \,^c\, t \cdot ((t \gg x) \parallel y)) \ll y \\
(37) \approx\ & (b \,^c\, t \cdot t \ggg ((t \gg x) \parallel y)) \ll y \\
(20) \approx\ & (b \,^c\, t \cdot ((t \ggg t \gg x) \parallel t \ggg y)) \ll y \\
\text{assumption, } (32) \approx\ & (b \,^c\, t \cdot (x \parallel t \ggg y)) \ll y \\
\text{(ATCC7)} \approx\ & (b \,^c\, t \ll y) \cdot (x \parallel t \ggg y)
\end{aligned}
$$

□

B.3 Simplifications of Normal Forms

In this appendix we show how combinations of terms in TPEGNF and terms with the syntax (6.3) can be simplified. We also prove correctness of these simplifications by proving that the equality of the initial and the simplified terms is derivable from the axioms of timed μCRL.

B.3.1 Definitions of *simpl* and *simpl′*

The partial functions *simpl* and *simpl′* that are used for the definition of function *guard* in Section 4.2.4 are defined as follows.

Sequential composition

$$
\begin{aligned}
\mathit{simpl}\Bigg(\Big(&\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,^c\, t_i \cdot p_i \lhd c_i \rhd \delta \,^c\, \mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,^c\, t_j \lhd c_j \rhd \delta \,^c\, \mathbf{0} \\
&+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,^c\, t_\delta \lhd c_\delta \rhd \delta \,^c\, \mathbf{0}\Big) \cdot p\Bigg) \\
&= \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,^c\, t_i \cdot (p_i \cdot p) \lhd c_i \rhd \delta \,^c\, \mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,^c\, t_j \cdot p \lhd c_j \rhd \delta \,^c\, \mathbf{0} \\
&+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,^c\, t_\delta \lhd c_\delta \rhd \delta \,^c\, \mathbf{0}
\end{aligned}
$$

Left merge

$$
\begin{aligned}
\mathit{simpl}'\Bigg(\Big(&\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,^c\, t_i \cdot p_i \lhd c_i \rhd \delta \,^c\, \mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,^c\, t_j \lhd c_j \rhd \delta \,^c\, \mathbf{0} \\
&+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,^c\, t_\delta \lhd c_\delta \rhd \delta \,^c\, \mathbf{0}\Big) \mathbin{\|\!\_} p, p'\Bigg)
\end{aligned}
$$

$$= \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} (\mathsf{a}_i(\overrightarrow{t_i}) \,{}^{\varsigma}\, t_i \ll p') \cdot ((t_i \gg p_i) \parallel p) \lhd c_i \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} (\mathsf{a}_j(\overrightarrow{t_j}) \,{}^{\varsigma}\, t_j \ll p') \cdot p \lhd c_j \rhd \delta{}^{\varsigma}\mathbf{0} + \sum_{\overrightarrow{e_\delta:E_\delta}} (\delta \,{}^{\varsigma}\, t_\delta \ll p') \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0}$$

Communication merge

$$\mathit{simpl}\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) \,{}^{\varsigma}\, t_i(\overrightarrow{d,e_i}) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \,{}^{\varsigma}\, t_j(\overrightarrow{d,e_i}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t_\delta(\overrightarrow{d,e_i}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0}\Big)$$
$$\mid \Big(\sum_{i\in I'}\sum_{\overrightarrow{e'_i:E'_i}} \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d',e'_i})) \,{}^{\varsigma}\, t'_i(\overrightarrow{d',e'_i}) \cdot p'_i(\overrightarrow{d',e'_i}) \lhd c'_i(\overrightarrow{d',e'_i}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{j\in J'}\sum_{\overrightarrow{e'_j:E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d',e'_j})) \,{}^{\varsigma}\, t'_j(\overrightarrow{d',e'_i}) \lhd c'_j(\overrightarrow{d',e'_j}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e'_\delta:E'_\delta}} \delta \,{}^{\varsigma}\, t'_\delta(\overrightarrow{d',e'_i}) \lhd c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta{}^{\varsigma}\mathbf{0}\Big)\Bigg)$$
$$= \sum_{(k,l)\in I\gamma I'}\ \sum_{\overrightarrow{e_k:E_k,e'_l:E'_l}} \gamma(\mathsf{a}_k,\mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \,{}^{\varsigma}\, t_k(\overrightarrow{d,e_k}) \cdot (p_k(\overrightarrow{d,e_k}) \parallel p_l(\overrightarrow{d',e'_l}))$$
$$\lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_k}) \wedge \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{(k,l)\in I\gamma J'}\ \sum_{\overrightarrow{e_k:E_k,e'_l:E'_l}} \gamma(\mathsf{a}_k,\mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \,{}^{\varsigma}\, t_k(\overrightarrow{d,e_k}) \cdot p_k(\overrightarrow{d,e_k})$$
$$\lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_k}) \wedge \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{(k,l)\in J\gamma I'}\ \sum_{\overrightarrow{e_k:E_k,e'_l:E'_l}} \gamma(\mathsf{a}_k,\mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \,{}^{\varsigma}\, t_k(\overrightarrow{d,e_k}) \cdot p_l(\overrightarrow{d',e'_l})$$
$$\lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_k}) \wedge \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{(k,l)\in J\gamma J'}\ \sum_{\overrightarrow{e_k:E_k,e'_l:E'_l}} \gamma(\mathsf{a}_k,\mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \,{}^{\varsigma}\, t_k(\overrightarrow{d,e_k})$$
$$\lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_k}) \wedge \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \wedge c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta{}^{\varsigma}\mathbf{0}$$
$$+ \sum_{k\in I\cup J\cup\{\delta\}}\ \sum_{l\in I'\cup J'\cup\{\delta\}}\ \sum_{\overrightarrow{e_k:E_k,e'_l:E'_l}} \delta{}^{\varsigma}\mathit{min}(t_k(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_k})) \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta{}^{\varsigma}\mathbf{0}$$

where $P\gamma Q = \{(p,q) \in P \times Q \mid \gamma(\mathsf{a}_p, \mathsf{a}'_q) \text{ is defined}\}$

We note that in case the function *simpl* is used for elimination of parallel composition, the last summand in the definition for communication merge can be omitted, because the summands obtained by the left merge elimination will contain them anyway.

"At" operation

$$\begin{aligned}
&simpl\Bigg(\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,\text{\textlangle}\, t_i \cdot p_i \lhd c_i \rhd \delta\text{\textlangle}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,\text{\textlangle}\, t_j \lhd c_j \rhd \delta\text{\textlangle}\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,\text{\textlangle}\, t_\delta \lhd c_\delta \rhd \delta\text{\textlangle}\mathbf{0}\Big) \,\text{\textlangle}\, t\Bigg) \\
&= \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,\text{\textlangle}\, t_i \cdot p_i \lhd t = t_i \wedge c_i \rhd \delta\text{\textlangle}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,\text{\textlangle}\, t_j \lhd t = t_j \wedge c_j \rhd \delta\text{\textlangle}\mathbf{0} \\
&\quad + \sum_{k\in I\cup J\cup\{\delta\}}\sum_{\overrightarrow{e_k:E_k}} \delta \,\text{\textlangle}\, min(t, t_k) \lhd c_k \rhd \delta\text{\textlangle}\mathbf{0}
\end{aligned}$$

Weak time initialization

$$\begin{aligned}
&simpl\Bigg(t \ggg \Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,\text{\textlangle}\, t_i \cdot p_i \lhd c_i \rhd \delta\text{\textlangle}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,\text{\textlangle}\, t_j \lhd c_j \rhd \delta\text{\textlangle}\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,\text{\textlangle}\, t_\delta \lhd c_\delta \rhd \delta\text{\textlangle}\mathbf{0}\Big)\Bigg) \\
&= \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,\text{\textlangle}\, t_i \cdot p_i \lhd t \leq t_i \wedge c_i \rhd \delta\text{\textlangle}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,\text{\textlangle}\, t_j \lhd t \leq t_j \wedge c_j \rhd \delta\text{\textlangle}\mathbf{0} \\
&\quad + \sum_{k\in I\cup J\cup\{\delta\}}\sum_{\overrightarrow{e_k:E_k}} \delta \,\text{\textlangle}\, t_k \lhd c_k \rhd \delta\text{\textlangle}\mathbf{0}
\end{aligned}$$

B.3.2 Derivability Proofs for *simpl* and *simpl′*.

We have to prove that any result of *simpl*(p) is derivably equal to p provided that p is of the correct form. We consider the cases of the definition of *simpl* one by one.

Sequential composition For the sequential composition we have the following derivation. First we distribute sequential composition over $+$, $\sum$ and conditionals using the axioms (A4), (SUM5) and (Cond6T). After that we apply the axiom (A5) and (Lemma B.2.3.17) to obtain the right form.

$$\begin{aligned}
&\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \,\text{\textlangle}\, t_i \cdot p_i \lhd c_i \rhd \delta\text{\textlangle}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \,\text{\textlangle}\, t_j \lhd c_j \rhd \delta\text{\textlangle}\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,\text{\textlangle}\, t_\delta \lhd c_\delta \rhd \delta\text{\textlangle}\mathbf{0}\Big) \cdot p
\end{aligned}$$

$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) ^{\varsigma}\, t_i \cdot p_i\big) \cdot p \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) ^{\varsigma}\, t_j \cdot p \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta ^{\varsigma}\, t_\delta \cdot p \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}$$
$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) ^{\varsigma}\, t_i \cdot \big(p_i \cdot p\big) \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) ^{\varsigma}\, t_j \cdot p \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta ^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}$$

Left merge For the left merge case we have the following derivation. First we distribute the left merge over $+$, $\sum$ and conditionals using the axioms (CM4), (SUM6) and (Cond8T). After that we apply (CM3T), (CM2T) and Lemma B.2.4.36 to obtain the right form.

$$\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) ^{\varsigma}\, t_i \cdot p_i \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) ^{\varsigma}\, t_j \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta ^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}\Big) \mathbin{\|\!\_}\, p$$
$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) ^{\varsigma}\, t_i \cdot p_i\big) \mathbin{\|\!\_}\, p \lhd c_i \rhd \delta^{\varsigma}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) ^{\varsigma}\, t_j \mathbin{\|\!\_}\, p \lhd c_j \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta ^{\varsigma}\, t_\delta \mathbin{\|\!\_}\, p \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}$$
$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) ^{\varsigma}\, t_i \ll p\big) \cdot \big((t_i \gg p_i) \parallel p\big) \lhd c_i \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \big(\mathsf{a}_j(\overrightarrow{t_j}) ^{\varsigma}\, t_j \ll p\big) \cdot p \lhd c_j \rhd \delta^{\varsigma}\mathbf{0} + \sum_{\overrightarrow{e_\delta:E_\delta}} \big(\delta ^{\varsigma}\, t_\delta \ll p\big) \lhd c_\delta \rhd \delta^{\varsigma}\mathbf{0}$$

Now, if for some process term p', $p \approx p'$ is derivable, then $simpl'(P \mathbin{\|\!\_}\, p, p') \approx P \mathbin{\|\!\_}\, p$ is also derivable from the axioms of timed μCRL.

Communication merge For case of communication merge we have the following derivation.

$$\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) ^{\varsigma}\, t_i(\overrightarrow{d,e_i}) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) ^{\varsigma}\, t_j(\overrightarrow{d,e_i}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta^{\varsigma}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta ^{\varsigma}\, t_\delta(\overrightarrow{d,e_i}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta^{\varsigma}\mathbf{0} \quad \Big)$$

$$
\begin{aligned}
|\Big(\sum_{i\in I'}\sum_{\overrightarrow{e'_i:E'_i}} & \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d',e'_i})) \cdot t'_i(\overrightarrow{d',e'_i}) \cdot p'_i(\overrightarrow{d',e'_i}) \lhd c'_i(\overrightarrow{d',e'_i}) \rhd \delta \cdot \mathbf{0} \\
& + \sum_{j\in J'}\sum_{\overrightarrow{e'_j:E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d',e'_j})) \cdot t'_j(\overrightarrow{d',e'_i}) \lhd c'_j(\overrightarrow{d',e'_j}) \rhd \delta \cdot \mathbf{0} \\
& + \sum_{\overrightarrow{e'_\delta:E'_\delta}} \delta \cdot t'_\delta(\overrightarrow{d',e'_i}) \lhd c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0}\Big)
\end{aligned}
$$

First we distribute the communication over $+$, $\sum$ and conditionals using the axioms (CM8), (CM9), (SUM7), (SUM7′), (Cond9T) and (Cond9′). After that we apply the axioms (ATA3), (ATA7), (ATA8), (CM7), (CM5), and (CM6). In order to avoid ambiguities we rename the index variables to k and l, with the first one ranging over I and J, and the second one ranging over I' and J'.

$$
\begin{aligned}
\approx & \sum_{k\in I}\sum_{l\in I'}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E'_l}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \\
& \qquad \cdot (p_k(\overrightarrow{d,e_k}) \parallel p'_l(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in I}\sum_{l\in J'}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E'_l}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \cdot p_k(\overrightarrow{d,e_k}) \\
& \qquad \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in I}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_\delta:E'_\delta}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \delta) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_\delta(\overrightarrow{d',e'_l}) \cdot p_k(\overrightarrow{d,e_k}) \\
& \qquad \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J}\sum_{l\in I'}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E'_l}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \cdot p'_l(\overrightarrow{d',e'_l}) \\
& \qquad \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J}\sum_{l\in J'}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_l:E'_l}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \\
& \qquad \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J}\sum_{\overrightarrow{e_k:E_k}}\sum_{\overrightarrow{e'_\delta:E'_\delta}} (\mathsf{a}_k(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \mid \delta) \cdot t_k(\overrightarrow{d,e_k}) \cdot t'_\delta(\overrightarrow{d',e'_l}) \lhd c_k(\overrightarrow{d,e_k}) \wedge c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{l\in I'}\sum_{\overrightarrow{e_\delta:E_\delta}}\sum_{\overrightarrow{e'_l:E'_l}} (\delta \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_\delta(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \cdot p'_l(\overrightarrow{d',e'_l}) \\
& \qquad \lhd c_\delta(\overrightarrow{d,e_\delta}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{l\in J'}\sum_{\overrightarrow{e_\delta:E_\delta}}\sum_{\overrightarrow{e'_l:E'_l}} (\delta \mid \mathsf{a}'_l(\overrightarrow{f'_l}(\overrightarrow{d',e'_l}))) \cdot t_\delta(\overrightarrow{d,e_k}) \cdot t'_l(\overrightarrow{d',e'_l}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \wedge c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{\overrightarrow{e_\delta:E_\delta}}\sum_{\overrightarrow{e'_\delta:E'_\delta}} (\delta \mid \delta) \cdot t_\delta(\overrightarrow{d,e_k}) \cdot t'_\delta(\overrightarrow{d',e'_l}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \wedge c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0}
\end{aligned}
$$

Now we note that, according to Lemma B.2.3.35, all of the communications where δ takes part, are equal to $\delta \cdot min(t_k, t'_l)$. For the remaining four summands we apply Lemma B.2.3.33&.34. In the result those summands are split in two parts each, one where the actions communicate (indices are taken from $I\gamma J$ sets), and the other one where the communication results into δ.

$$
\begin{aligned}
\approx & \sum_{(k,l)\in I\gamma J} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \cdot t_k(\overrightarrow{d,e_k}) \cdot (p_k(\overrightarrow{d,e_k}) \parallel p'_l(\overrightarrow{d',e'_l})) \\
& \qquad \lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_l}) \land \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \land c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in I} \sum_{l\in J} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{(k,l)\in I\gamma J'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \cdot t_k(\overrightarrow{d,e_k}) \cdot p_k(\overrightarrow{d,e_k}) \\
& \qquad \lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_l}) \land \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \land c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in I} \sum_{l\in J'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in I} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_\delta:E'_\delta}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_\delta(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{(k,l)\in J\gamma I'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \cdot t'_l(\overrightarrow{d',e'_l}) \cdot p'_l(\overrightarrow{d',e'_l}) \\
& \qquad \lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_l}) \land \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \land c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J} \sum_{l\in I'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{(k,l)\in J\gamma J'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d,e_k})) \cdot t'_l(\overrightarrow{d',e'_l}) \\
& \qquad \lhd t_k(\overrightarrow{d,e_k}) = t'_l(\overrightarrow{d',e'_l}) \land \overrightarrow{f_k}(\overrightarrow{d,e_k}) = \overrightarrow{f'_l}(\overrightarrow{d',e'_l}) \land c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J} \sum_{l\in J'} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{k\in J} \sum_{\overrightarrow{e_k:E_k}} \sum_{\overrightarrow{e'_\delta:E'_\delta}} \delta \cdot min(t_k(\overrightarrow{d,e_k}), t'_\delta(\overrightarrow{d',e'_l})) \lhd c_k(\overrightarrow{d,e_k}) \land c'_\delta(\overrightarrow{d',e'_\delta}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{l\in I'} \sum_{\overrightarrow{e_\delta:E_\delta}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_\delta(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_\delta(\overrightarrow{d,e_\delta}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0} \\
+ & \sum_{l\in J'} \sum_{\overrightarrow{e_\delta:E_\delta}} \sum_{\overrightarrow{e'_l:E'_l}} \delta \cdot min(t_\delta(\overrightarrow{d,e_k}), t'_l(\overrightarrow{d',e'_l})) \lhd c_\delta(\overrightarrow{d,e_\delta}) \land c'_l(\overrightarrow{d',e'_l}) \rhd \delta \cdot \mathbf{0}
\end{aligned}
$$

$$+ \sum_{\overrightarrow{e_\delta : E_\delta}} \sum_{\overrightarrow{e'_\delta : E'_\delta}} \delta \text{ ‹ } min(t_\delta(\overrightarrow{d, e_k}) \text{ ‹ } t'_\delta(\overrightarrow{d', e'_l})) \lhd c_\delta(\overrightarrow{d, e_\delta}) \wedge c'_\delta(\overrightarrow{d', e'_\delta}) \rhd \delta \text{‹} \mathbf{0}$$

Finally we use the unions of index sets to combine all nine summands with δ into one.

$$\begin{aligned}
\approx & \sum_{(k,l) \in I\gamma J} \sum_{\overrightarrow{e_k : E_k}} \sum_{\overrightarrow{e'_l : E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d, e_k})) \text{ ‹ } t_k(\overrightarrow{d, e_k}) \cdot (p_k(\overrightarrow{d, e_k}) \parallel p'_l(\overrightarrow{d', e'_l})) \\
& \qquad \lhd t_k(\overrightarrow{d, e_k}) = t'_l(\overrightarrow{d', e'_l}) \wedge \overrightarrow{f_k}(\overrightarrow{d, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d', e'_l}) \wedge c_k(\overrightarrow{d, e_k}) \wedge c'_l(\overrightarrow{d', e'_l}) \rhd \delta \text{‹} \mathbf{0} \\
+ & \sum_{(k,l) \in I\gamma J'} \sum_{\overrightarrow{e_k : E_k}} \sum_{\overrightarrow{e'_l : E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d, e_k})) \text{ ‹ } t_k(\overrightarrow{d, e_k}) \cdot p_k(\overrightarrow{d, e_k}) \\
& \qquad \lhd t_k(\overrightarrow{d, e_k}) = t'_l(\overrightarrow{d', e'_l}) \wedge \overrightarrow{f_k}(\overrightarrow{d, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d', e'_l}) \wedge c_k(\overrightarrow{d, e_k}) \wedge c'_l(\overrightarrow{d', e'_l}) \rhd \delta \text{‹} \mathbf{0} \\
+ & \sum_{(k,l) \in J\gamma I'} \sum_{\overrightarrow{e_k : E_k}} \sum_{\overrightarrow{e'_l : E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d, e_k})) \text{ ‹ } t'_l(\overrightarrow{d', e'_l}) \cdot p'_l(\overrightarrow{d', e'_l}) \\
& \qquad \lhd t_k(\overrightarrow{d, e_k}) = t'_l(\overrightarrow{d', e'_l}) \wedge \overrightarrow{f_k}(\overrightarrow{d, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d', e'_l}) \wedge c_k(\overrightarrow{d, e_k}) \wedge c'_l(\overrightarrow{d', e'_l}) \rhd \delta \text{‹} \mathbf{0} \\
+ & \sum_{(k,l) \in J\gamma J'} \sum_{\overrightarrow{e_k : E_k}} \sum_{\overrightarrow{e'_l : E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d, e_k})) \text{ ‹ } t'_l(\overrightarrow{d', e'_l}) \\
& \qquad \lhd t_k(\overrightarrow{d, e_k}) = t'_l(\overrightarrow{d', e'_l}) \wedge \overrightarrow{f_k}(\overrightarrow{d, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d', e'_l}) \wedge c_k(\overrightarrow{d, e_k}) \wedge c'_l(\overrightarrow{d', e'_l}) \rhd \delta \text{‹} \mathbf{0} \\
+ & \sum_{k \in I \cup J \cup \{\delta\}} \sum_{l \in I' \cup J' \cup \{\delta\}} \sum_{\overrightarrow{e_k : E_k}} \sum_{\overrightarrow{e'_l : E'_l}} \delta \text{‹} min(t_k(\overrightarrow{d, e_k}), t'_l(\overrightarrow{d', e'_l})) \lhd c_k(\overrightarrow{d, e_k}) \wedge c'_l(\overrightarrow{d', e'_l}) \rhd \delta \text{‹} \mathbf{0}
\end{aligned}$$

"At"-operation For the case of "at"-operation we have the following derivation. First we distribute the "at"-operation over $+$, $\sum$ conditionals, and sequential composition using the axioms (ATA4), (ATA5′) and (ATA3). After that we apply the axiom (ATA1′) and Lemma B.2.3.22 to reduce double "at"-operations. Next, we apply the axioms (Cond7T), (Cond4T) and (SUM4) to lift the $+$ to the uppermost level and reduce double conditionals. Finally, we apply Lemma B.2.3.17 to get rid of p_i in $\delta \text{ ‹ } min(t_i, t) \cdot p_i$ and combine three sums with $\delta \text{ ‹ } min(t_k, t)$ into one.

$$\begin{aligned}
& \Big(\sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \text{ ‹ } t_i \cdot p_i \lhd c_i \rhd \delta \text{‹} \mathbf{0} + \sum_{j \in J} \sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \text{ ‹ } t_j \lhd c_j \rhd \delta \text{‹} \mathbf{0} \\
& \quad + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \text{ ‹ } t_\delta \lhd c_\delta \rhd \delta \text{‹} \mathbf{0} \Big) \text{ ‹ } t \\
\approx & \sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \text{ ‹ } t_i \text{ ‹ } t \cdot p_i \lhd c_i \rhd \delta \text{‹} \mathbf{0} + \sum_{j \in J} \sum_{\overrightarrow{e_j : E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \text{ ‹ } t_j \text{ ‹ } t \lhd c_j \rhd \delta \text{‹} \mathbf{0} \\
& \quad + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \text{ ‹ } t_\delta \text{ ‹ } t \lhd c_\delta \rhd \delta \text{‹} \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
&\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \lhd eq(t_i,t) \rhd \delta\cdot\mathbf{0} + \delta \cdot min(t_i,t)\big) \cdot p_i \lhd c_i \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \big(\mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd eq(t_j,t) \rhd \delta\cdot\mathbf{0} + \delta \cdot min(t_j,t)\big) \lhd c_j \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot min(t_\delta,t) \lhd c_\delta \rhd \delta\cdot\mathbf{0} \\
&\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \lhd eq(t_i,t) \wedge c_i \rhd \delta\cdot\mathbf{0} + \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \delta \cdot min(t_i,t) \cdot p_i \lhd c_i \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd eq(t_j,t) \wedge c_j \rhd \delta\cdot\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \delta \cdot min(t_j,t) \lhd c_j \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot min(t_\delta,t) \lhd c_\delta \rhd \delta\cdot\mathbf{0} \\
&\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i})\cdot t_i \lhd eq(t_i,t) \wedge c_i \rhd \delta\cdot\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j})\cdot t_j \lhd eq(t_j,t) \wedge c_j \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{k\in I\cup J\cup\{\delta\}}\sum_{\overrightarrow{e_k:E_k}} \delta \cdot min(t_k,t) \lhd c_k \rhd \delta\cdot\mathbf{0}
\end{aligned}$$

Weak time initialization For the case of weak time initialization we have the following derivation. First we distribute the operation over $+$, $\sum$, conditionals, and sequential composition, using Lemma B.2.4.6,.12,.13&.7. After that we apply Lemma B.2.4.2&.3 to eliminate weak time initialization from time stamped actions or δ. Next, we apply the axioms (Cond7T), (Cond4T) and (SUM4) to lift the $+$ to the uppermost level and reduce double conditionals. Finally, we apply Lemma B.2.3.17 to get rid of p_i in $\delta \cdot min(t_i,t)\cdot p_i$ and combine three sums with $\delta \cdot min(t_k,t)$ into one.

$$\begin{aligned}
t \ggg &\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \cdot p_i \lhd c_i \rhd \delta\cdot\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd c_j \rhd \delta\cdot\mathbf{0} \\
&\qquad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta\cdot\mathbf{0}\Big) \\
&\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} (t \ggg \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i) \cdot p_i \lhd c_i \rhd \delta\cdot\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} (t \ggg \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j) \lhd c_j \rhd \delta\cdot\mathbf{0} \\
&\qquad + \sum_{\overrightarrow{e_\delta:E_\delta}} (t \ggg \delta \cdot t_\delta) \lhd c_\delta \rhd \delta\cdot\mathbf{0} \\
&\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \big(\mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \lhd t \le t_i \rhd \delta\cdot\mathbf{0} + \delta \cdot t_i\big) \cdot p_i \lhd c_i \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \big(\mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd t \le t_j \rhd \delta\cdot\mathbf{0} + \delta \cdot t_j\big) \lhd c_j \rhd \delta\cdot\mathbf{0} \\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta\cdot\mathbf{0}
\end{aligned}$$

$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \lhd t \le t_i \wedge c_i \rhd \delta^{\cdot}\mathbf{0} + \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \delta \cdot t_i \cdot p_i \lhd c_i \rhd \delta^{\cdot}\mathbf{0}$$
$$+ \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd t \le t_j \wedge c_j \rhd \delta^{\cdot}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \delta \cdot t_j \lhd c_j \rhd \delta^{\cdot}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta^{\cdot}\mathbf{0}$$
$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \lhd t \le t_i \wedge c_i \rhd \delta^{\cdot}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd t \le t_j \wedge c_j \rhd \delta^{\cdot}\mathbf{0}$$
$$+ \sum_{k\in I\cup J\cup\{\delta\}}\sum_{\overrightarrow{e_k:E_k}} \delta \cdot t_k \lhd c_k \rhd \delta^{\cdot}\mathbf{0}$$

B.3.3 Derivability Proof for *simpl1*.

Proposition B.3.1. *For any term p having the form of right-hand side of a TPEGNF equation the transformation performed by function simpl1 is derivable from the axioms of timed μCRL.*

Proof. First we apply Lemmas B.2.3.5,.8&.9 and B.2.3.6,.24&.4 to eliminate $\partial\mathcal{U}$, and then we combine three sets of summands into one.

$$\partial\mathcal{U}\Big(\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{t_i}) \cdot t_i \cdot p_i \lhd c_i \rhd \delta^{\cdot}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{t_j}) \cdot t_j \lhd c_j \rhd \delta^{\cdot}\mathbf{0}$$
$$+ \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta^{\cdot}\mathbf{0}\Big)$$
$$\approx \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \delta \cdot t_i \lhd c_i \rhd \delta^{\cdot}\mathbf{0} + \sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \delta \cdot t_j \lhd c_j \rhd \delta^{\cdot}\mathbf{0} + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta^{\cdot}\mathbf{0}$$
$$\approx \sum_{k\in I\cup J\cup\{\delta\}}\sum_{\overrightarrow{e_k:E_k}} \delta \cdot t_k \lhd c_k \rhd \delta^{\cdot}\mathbf{0}$$

□

The following proposition shows how several $\delta \cdot t$ summands can be combined.

Proposition B.3.2. *If u:Time is not free in $t_\delta(\overrightarrow{d,e_\delta})$ and $c_\delta(\overrightarrow{d,e_\delta})$, then*

$$\sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta(\overrightarrow{d,e_\delta}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta^{\cdot}\mathbf{0} \approx \sum_{\overrightarrow{u,e_\delta:Time,E_\delta}} \delta \cdot u \lhd u \le t_\delta(\overrightarrow{d,e_\delta}) \wedge c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta^{\cdot}\mathbf{0}$$

If for every $i \in \{0,\dots,n\}$, u:Time is not free in $t_i(\overrightarrow{d,e_i})$ and $c_i(\overrightarrow{d,e_i})$, then

$$\sum_{i\in\{0,\dots,n\}}\sum_{\overrightarrow{e_i:E_i}} \delta \cdot t_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta^{\cdot}\mathbf{0}$$
$$\approx \sum_{u:Time}\sum_{\overrightarrow{e_0:E_0}}\cdots\sum_{\overrightarrow{e_n:E_n}} \delta \cdot u \lhd \bigvee\nolimits_{0\le i\le n}\big(u \le t_i(\overrightarrow{d,e_i}) \wedge c_i(\overrightarrow{d,e_i})\big) \rhd \delta^{\cdot}\mathbf{0}$$

Proof.

$$\sum_{\overrightarrow{u,e_\delta:Time,E_\delta}} \delta \cdot u \lhd u \leq t_\delta(\overrightarrow{d,e_\delta}) \wedge c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta \cdot \mathbf{0}$$

$$\begin{aligned}
&(\text{Lemma 2.2.5}),\\
(\text{Cond4T}), (\text{SUM12T}) &\approx \sum_{\overrightarrow{e_\delta:E_\delta}} \Big(\sum_{u:Time} \delta \cdot u \lhd u \leq t_\delta(\overrightarrow{d,e_\delta}) \rhd \delta \cdot \mathbf{0} \Big) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta \cdot \mathbf{0}\\
(\text{Lemma B.2.3.23}) &\approx \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta(\overrightarrow{d,e_\delta}) \lhd c_\delta(\overrightarrow{d,e_\delta}) \rhd \delta \cdot \mathbf{0}
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{u:Time} \sum_{\overrightarrow{e_0:E_0}} \cdots \sum_{\overrightarrow{e_n:E_n}} \delta \cdot u \lhd \bigvee\nolimits_{0 \leq i \leq n} \big(u \leq t_i(\overrightarrow{d,e_i}) \wedge c_i(\overrightarrow{d,e_i})\big) \rhd \delta \cdot \mathbf{0}\\
(\text{Cond5T}) &\approx \sum_{u:Time} \sum_{\overrightarrow{e_0:E_0}} \cdots \sum_{\overrightarrow{e_n:E_n}} \sum_{i \in \{0,1,\cdots,n\}} \delta \cdot u \lhd u \leq t_i(\overrightarrow{d,e_i}) \wedge c_i(\overrightarrow{d,e_i}) \rhd \delta \cdot \mathbf{0}\\
(\text{SUM4}) &\approx \sum_{i \in \{0,1,\cdots,n\}} \sum_{u:Time} \sum_{\overrightarrow{e_0:E_0}} \cdots \sum_{\overrightarrow{e_n:E_n}} \delta \cdot u \lhd u \leq t_i(\overrightarrow{d,e_i}) \wedge c_i(\overrightarrow{d,e_i}) \rhd \delta \cdot \mathbf{0}\\
(\text{SUM1}) &\approx \sum_{i \in \{0,1,\cdots,n\}} \sum_{u:Time} \sum_{\overrightarrow{e_i:E_i}} \delta \cdot u \lhd u \leq t_i(\overrightarrow{d,e_i}) \wedge c_i(\overrightarrow{d,e_i}) \rhd \delta \cdot \mathbf{0}\\
\text{above derivation} &\approx \sum_{i \in \{0,1,\cdots,n\}} \sum_{\overrightarrow{e_i:E_i}} \delta \cdot t_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta \cdot \mathbf{0}
\end{aligned}$$

□

B.3.4 Elimination of $\ll$ from *simpl'*

The following proposition shows how $\ll$ is eliminated from the terms obtained after applying *simpl* for the left merge elimination.

Proposition B.3.3. *If all of the variables in $\overrightarrow{e_\delta:E_\delta}$ are not free in the terms $\overrightarrow{t}$ and t, then*

$$\begin{aligned}
\mathsf{a}(\overrightarrow{t}) \cdot t \ll \Big(\sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta \cdot \mathbf{0}\Big) &\approx \sum_{\overrightarrow{e_\delta:E_\delta}} \mathsf{a}(\overrightarrow{t}) \cdot t \lhd eq(t, \mathbf{0}) \vee (t \leq t_\delta \wedge c_\delta) \rhd \delta \cdot \mathbf{0}\\
&\quad + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot min(t_\delta, t) \lhd c_\delta \rhd \delta \cdot \mathbf{0}\\
\delta \cdot t \ll \Big(\sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta \cdot \mathbf{0}\Big) &\approx \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \cdot min(t, t_\delta) \lhd c_\delta \rhd \delta \cdot \mathbf{0}
\end{aligned}$$

Proof.

$$
\begin{aligned}
& \mathsf{a}(\overrightarrow{t}) \cdot t \ll \Big(\sum_{\overrightarrow{e_\delta : E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta \cdot \mathbf{0}\Big) \\
\text{(ATC4), (ATC5}') \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} \Big(\big(\mathsf{a}(\overrightarrow{t}) \cdot t \ll \delta \cdot t_\delta\big) \lhd c_\delta \rhd \delta \cdot \mathbf{0} + \mathsf{a}(\overrightarrow{t}) \cdot t \cdot \mathbf{0}\Big) \\
\text{(Lemma B.2.4.34)}, \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} \Big(\big(\mathsf{a}(\overrightarrow{t}) \cdot t \lhd t \leq t_\delta \rhd \delta \cdot \mathbf{0} + \delta \cdot \mathit{min}(t_\delta, t)\big) \lhd c_\delta \rhd \delta \cdot \mathbf{0} \\
\text{(ATA1}') \quad & + \mathsf{a}(\overrightarrow{t}) \cdot t \lhd t = \mathbf{0} \rhd \delta \cdot \mathbf{0} + \delta \cdot \mathit{min}(t, \mathbf{0})\Big) \\
\text{(Cond7T), (Cond4T)}, \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} \Big(\mathsf{a}(\overrightarrow{t}) \cdot t \lhd t \leq t_\delta \wedge c_\delta \rhd \delta \cdot \mathbf{0} \\
\text{(Lemma B.2.1.21)}, \quad & + \delta \cdot \mathit{min}(t_\delta, t) \lhd c_\delta \rhd \delta \cdot \mathbf{0} \\
\text{(Lemma B.2.3.3)} \quad & + \mathsf{a}(\overrightarrow{t}) \cdot t \lhd t = \mathbf{0} \rhd \delta \cdot \mathbf{0}\Big) \\
\text{(Cond5T), (SUM4)} \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} \mathsf{a}(\overrightarrow{t}) \cdot t \lhd t = \mathbf{0} \vee (t \leq t_\delta \wedge c_\delta) \rhd \delta \cdot \mathbf{0} \\
& + \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \cdot \mathit{min}(t_\delta, t) \lhd c_\delta \rhd \delta \cdot \mathbf{0}
\end{aligned}
$$

and

$$
\begin{aligned}
& \delta \cdot t \ll \Big(\sum_{\overrightarrow{e_\delta : E_\delta}} \delta \cdot t_\delta \lhd c_\delta \rhd \delta \cdot \mathbf{0}\Big) \\
\text{(ATC4), (ATC5}') \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} (\delta \cdot t_\delta \ll \delta \cdot t) \lhd c_\delta \rhd \delta \cdot \mathbf{0} \\
\text{(Lemma B.2.4.35)} \approx\ & \sum_{\overrightarrow{e_\delta : E_\delta}} \delta \cdot \mathit{min}(t, t_\delta) \lhd c_\delta \rhd \delta \cdot \mathbf{0}
\end{aligned}
$$

□

B.3.5 Proof of Well-Timedness for *simpl2'*

Lemma B.3.4. *For every term q in the form of the right-hand side of TPEGNF, the term $(\mathsf{a}(\overrightarrow{t}) \cdot t \ll q) \cdot t \gg q$, and if $p \approx t \gg p$, then also the term $(\mathsf{a}(\overrightarrow{t}) \cdot t \ll q) \cdot (p \| t \gg q)$, can be transformed to a well-timed term.*

Proof. By Lemma B.2.3.15 we have that

$$\mathsf{a}(\overrightarrow{t}) \cdot t \ll q \approx \mathsf{a}(\overrightarrow{t}) \cdot t \ll \partial\mathcal{U}(q)$$

By Proposition B.3.1 we get that $\partial\mathcal{U}(q)$ has the form (for some $\overrightarrow{e_\delta{:}E_\delta}$ not free in $\overrightarrow{t}$ and t.)

$$\sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0}$$

Now, by Proposition B.3.3 we get that

$$\mathsf{a}(\overrightarrow{t}) \,{}^{\varsigma}\, t \ll q \approx \sum_{\overrightarrow{e_\delta:E_\delta}} \mathsf{a}(\overrightarrow{t}) \,{}^{\varsigma}\, t \lhd eq(t,\mathbf{0}) \vee (t \le t_\delta \wedge c_\delta) \rhd \delta{}^{\varsigma}\mathbf{0} + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, min(t_\delta, t) \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0}$$

and by the axioms (A4),(SUM5) and (Cond6T) and Lemma B.2.3.17 for any term r we get that

$$(\mathsf{a}(\overrightarrow{t}) {}^{\varsigma} t \ll q) \cdot r \approx \sum_{\overrightarrow{e_\delta:E_\delta}} \mathsf{a}(\overrightarrow{t}) {}^{\varsigma} t \cdot r \lhd eq(t,\mathbf{0}) \vee (t \le t_\delta \wedge c_\delta) \rhd \delta\mathbf{0} + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta {}^{\varsigma} min(t_\delta, t) \lhd c_\delta \rhd \delta\mathbf{0}$$

According to Definition 6.2.12 we need to show that $eq(t,\mathbf{0}) \vee (t \le t_\delta \wedge c_\delta) \approx \mathbf{t}$ implies $r \approx t \gg r$. For the case when $r = t \ggg q$ we need to show that $t \ggg q \approx t \gg t \ggg q$. Due to the fact that $t \gg t \ggg q \approx t \ggg q + \delta{}^{\varsigma}t$ (by axiom (ATB0) and Lemma B.2.4.18&.3 and Lemma B.2.1.24), it is enough to show that $t \ggg q \approx t \ggg q + \delta \,{}^{\varsigma}\, t$ is implied by $eq(t,0) \vee (t \le t_\delta \wedge c_\delta \approx \mathbf{t})$.

$$\begin{aligned}
& t \ggg q \\
\text{(A6T)} \approx\ & t \ggg (q + \partial\mathcal{U}(q)) \\
\text{(Lemma B.2.4.6\&.3)} \approx\ & t \ggg q + \partial\mathcal{U}(q) \\
\text{assumption} \approx\ & t \ggg q + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t_\delta \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Lemma B.2.3.23)} \approx\ & t \ggg q + \sum_{\overrightarrow{e_\delta:E_\delta}} \Big(\sum_{u:Time} \delta \,{}^{\varsigma}\, u \lhd u \le t_\delta \rhd \delta{}^{\varsigma}\mathbf{0}\Big) \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(SUM3)} \approx\ & t \ggg q + \sum_{\overrightarrow{e_\delta:E_\delta}} \Big(\sum_{u:Time} \delta \,{}^{\varsigma}\, u \lhd u \le t_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
& \qquad + \delta \,{}^{\varsigma}\, t \lhd t \le t_\delta \rhd \delta{}^{\varsigma}\mathbf{0}\Big) \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(SUM12T), (Cond7T),} \approx\ & t \ggg q + \sum_{\overrightarrow{e_\delta:E_\delta}} \Big(\sum_{u:Time} \delta \,{}^{\varsigma}\, u \lhd u \le t_\delta \rhd \delta{}^{\varsigma}\mathbf{0}\Big) \lhd c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Cond4T), (SUM4)} \quad & + \sum_{\overrightarrow{e_\delta:E_\delta}} \delta \,{}^{\varsigma}\, t \lhd t \le t_\delta \wedge c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{this derivation,(SUM3)} \approx\ & t \ggg q + \delta \,{}^{\varsigma}\, t \lhd t \le t_\delta \wedge c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Lemma B.2.3.3\&.1),(PET)} \approx\ & t \ggg q + \delta{}^{\varsigma}t \lhd t \le t_\delta \wedge c_\delta \rhd \delta{}^{\varsigma}\mathbf{0} + \delta{}^{\varsigma}t \lhd eq(t,\mathbf{0}) \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{(Cond5T)} \approx\ & t \ggg q + \delta \,{}^{\varsigma}\, t \lhd eq(t,\mathbf{0}) \vee (t \le t_\delta \wedge c_\delta) \rhd \delta{}^{\varsigma}\mathbf{0} \\
\text{assumption,(Cond1)} \approx\ & t \ggg q + \delta \,{}^{\varsigma}\, t
\end{aligned}$$

For the case when $r = (p \,\|\, t \ggg q)$ we need to show that $(p \,\|\, t \ggg q) \approx t \gg (p \,\|\, t \ggg q)$. By assumption of the lemma we have that $t \gg p \approx p$. By the previous derivation we have that $t \ggg q \approx t \gg t \ggg q$. By Lemma B.2.4.33 we get the desired identity. $\square$

Appendix C

μCRL Code of *LM* and *ML* Data Types

The source code is split into six parts (Figure C.1): two basic parts and four terminal parts corresponding to the cases with or without the renaming operations, and with handshaking or with multi-party communication. For each terminal part all of the parts it depends upon are needed (only once in case of multiple dependencies).

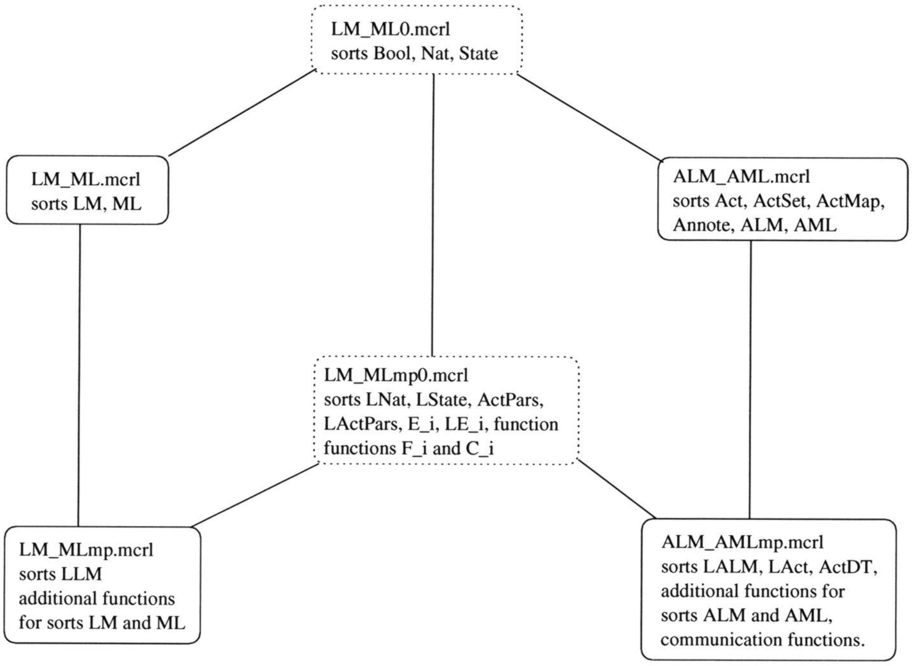

Figure C.1: Code Files Dependencies.

C.1 Basic Data Types

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% sorts Bool, Nat, State(generated)                          %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  Bool (Booleans)                                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort Bool
func
  T,F: -> Bool
map
  and: Bool#Bool      -> Bool
  or:  Bool#Bool      -> Bool
  not: Bool           -> Bool
  if:  Bool#Bool#Bool -> Bool
  eq:  Bool#Bool      -> Bool
  gt:  Bool#Bool      -> Bool
var
  b,b1,b2: Bool
rew
  and(T,b)=b                    and(b,T)=b
  and(b,F)=F                    and(F,b)=F
  and(b,b)=b
  and(b,not(b))=F               and(not(b),b)=F
  and(or(b,b1),b2)=or(and(b,b2),and(b1,b2))
  and(b,or(b1,b2))=or(and(b,b1),and(b,b2))

  or(T,b)=T                     or(b,T)=T
  or(b,F)=b                     or(F,b)=b
  or(b,b)=b
  or(b,not(b))=T                or(not(b),b)=T

  not(F)=T                      not(T)=F
  not(not(b))=b
  not(or(b,b1))=and(not(b),not(b1))
  not(and(b,b1))=or(not(b),not(b1))

  if(T,b1,b2)=b1                if(F,b1,b2)=b2
  if(b,b1,b1)=b1                if(not(b),b1,b2)=if(b,b2,b1)
  if(b,T,b2)=or(b,b2)           if(b,F,b2)=and(not(b),b2)
  if(b,b1,T)=or(not(b),b1)      if(b,b1,F)=and(b,b1)
  if(b,b1,b2)=or(or(and(b,b1),and(not(b),b2)),and(b1,b2))

  eq(b,b)=T eq(b,not(b))=F eq(not(b),b)=F eq(not(b),not(b1))=eq(b,b1)
  eq(F,b)=not(b) eq(b,F)=not(b) eq(T,b)=b eq(b,T)=b
  eq(b,b1)=or(and(b,b1),and(not(b),not(b1)))

  gt(b,b)=F gt(T,F)=T gt(b,T)=F

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort   Nat (Natural numbers with binary representations)  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort Nat
func
  0:                    -> Nat
  x2p1:          Nat -> Nat      % 2n+1
  x2p2:          Nat -> Nat      % 2n+2
map
  eq:            Nat#Nat        -> Bool
  1,2,3,4,5,6:                  -> Nat  % useful abbreviations
  x2p0:          Nat            -> Nat  % 2n
  succ:          Nat            -> Nat  % n+1
  gt:            Nat#Nat        -> Bool % greater than
  if:            Bool#Nat#Nat   -> Nat
  add,sub,csub: Nat#Nat         -> Nat  % addition, subtraction (partial), cut-off subtraction
  divides:       Nat#Nat        -> Bool % does the first argument divide the second? (partial)
```

```
var
  n,m: Nat b:Bool
rew
  gt(n,n)=F gt(0,n)=F gt(x2p1(n),0)=T gt(x2p2(n),0)=T

  gt(x2p1(n),x2p2(m))=gt(n,m)
  gt(x2p2(n),x2p1(m))=not(gt(m,n))

  gt(x2p1(n),x2p1(m))=gt(n,m)
  gt(x2p2(n),x2p2(m))=gt(n,m)

  % eq(n,m)=not(or(gt(n,m),gt(m,n)))     % sane, but inefficient

  eq(n,n)=T
  eq(x2p1(n),0)=F
  eq(0,x2p1(n))=F
  eq(x2p2(n),0)=F
  eq(0,x2p2(n))=F
  eq(x2p1(n),x2p2(m))=F
  eq(x2p2(n),x2p1(m))=F
  eq(x2p2(n),x2p2(m))=eq(n,m)
  eq(x2p1(n),x2p1(m))=eq(n,m)

  1=x2p1(0)  2=x2p2(0)                 % 1=2*0+1 2=2*0+2
  3=x2p1(1)  4=x2p2(1)                 % 3=2*1+1 4=2*1+2
  5=x2p1(2)  6=x2p2(2)                 % 5=2*2+1 6=2*2+2

  x2p0(0)=0                            % 2*0=0
  x2p0(x2p1(n))=x2p2(x2p0(n))          % 2(2n+1)=2(2n)+2
  x2p0(x2p2(n))=x2p2(x2p1(n))          % 2(2n+2)=2((2n+1)+1)=2(2n+1)+2

  succ(0)=x2p1(0)                      % 0+1=2*0+1
  succ(x2p1(n))=x2p2(n)                % (2n+1)+1=2n+2
  succ(x2p2(n))=x2p1(succ(n))          % (2n+2)+1=2(n+1)+1

  add(0,n)=n add(n,0)=n
  add(x2p1(n),x2p1(m))=x2p2(add(n,m))            % (2n+1)+(2m+1)=2(n+m)+2
  add(x2p2(n),x2p2(m))=x2p2(succ(add(n,m)))      % (2n+2)+(2m+2)=2(n+m)+4=2(n+m+1)+2
  add(x2p1(n),x2p2(m))=x2p1(succ(add(n,m)))      % (2n+1)+(2m+2)=2(n+m)+3=2(n+m+1)+1
  add(x2p2(n),x2p1(m))=x2p1(succ(add(n,m)))      % (2n+2)+(2m+1)=2(n+m)+3=2(n+m+1)+1

  sub(n,0)=n sub(n,n)=0                          % sub(0,x2p{1,2}) is undefined
  sub(x2p1(n),x2p1(m))=x2p0(sub(n,m))            % (2n+1)-(2m+1)=2(n-m)
  sub(x2p2(n),x2p2(m))=x2p0(sub(n,m))            % (2n+2)-(2m+2)=the same
  sub(x2p1(n),x2p2(m))=x2p1(sub(n,succ(m)))      % (2n+1)-(2m+2)=2(n-m)-1=2(n-(m+1))+1 --
                                                 % undefined if n=m!
  sub(x2p2(n),x2p1(m))=x2p1(sub(n,m))            % (2n+2)-(2m+1)=2(n-m)+1

  csub(n,m)=if(gt(n,m),sub(n,m),0)

  divides(x2p1(n),0)=T divides(x2p2(n),0)=T      % any n>0 divides 0; divides(0,n) is undef.
  divides(x2p1(n),x2p1(m))=                      % n divides m whenever it divides m-n
       and(not(gt(n,m)),divides(x2p1(n),sub(x2p1(m),x2p1(n))))
  divides(x2p1(n),x2p2(m))=
       and(not(gt(n,m)),divides(x2p1(n),sub(x2p2(m),x2p1(n))))
  divides(x2p2(n),x2p1(m))=F                     % even never divides odd.
  divides(x2p2(n),x2p2(m))=divides(succ(n),succ(m)) % (2n+2)|(2m+2) iff (n+1)|(m+1)

  if(T,n,m)=n if(F,n,m)=m if(b,n,n)=n if(not(b),n,m)=if(b,m,n)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  To be generated from the spec                              %%%
%%%%  The parts that do not parse before actual generation        %%%
%%%%  are commented out                                           %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  State (pre-LPE process parameters tuple)             %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
sort State
% func
  % state:D_0#...#D_n->State
map
  eq: State#State->Bool
  gt: State#State->Bool
  if: Bool#State#State->State
  % pr_0:State->D_0 ... pr_n:State->D_n
var
  d,e:State b:Bool
rew
  if(T,d,e)=d if(F,d,e)=e if(b,d,d)=d if(not(b),d,e)=if(b,e,d)
  % gt(state(d0,...,dn),state(e0,...,en))=
  %   or(gt(d0,e0),and(eq(d0,e0),...or(gt(d{n-1},e{n-1}),and(eq(d{n-1},e{n-1}),gt(dn,en)))...))
  eq(d,d)=T
  % eq(state(d0,...,dn),state(e0,...,en))=and(eq(d0,e0),...,and(eq(dn,en))...)
```

C.2 Handshaking *LM* and *ML* Data Types

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  LM And ML data types                                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  LM                                                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LM                        % list of ML or State elements
func
  LM0: ->LM                    % empty list
  seq1: State#LM->LM           % add one State element to the head of the list
  seqM: ML#LM->LM              % add one ML to the head of the list
                               %(first argument never ML(x))
map
  eq: LM#LM->Bool              % equality on LM
  if: Bool#LM#LM->LM

  gt: LM#LM->Bool
  conc: LM#LM->LM              % concatenate 2 LMs in a wf way.
  conp: ML#LM->LM              % prepend an ML to an LM

  lenf:LM->Nat                 % number of "ready" components
  getf1:LM#Nat->State          % get n-th component
  replf1:LM#Nat#LM->LM         % replace n-th component
  remf1:LM#Nat->LM             % remove n-th component
  replf2:LM#Nat#Nat#LM#LM->LM  % replace n-th and m-th components
  replremf2:LM#Nat#Nat#LM->LM  % replace n-th and remove m-th components
  remf2:LM#Nat#Nat->LM         % remove n-th and m-th components

  parc: LM#LM->LM              % compose 2 LMs parallely
  seqc: LM#LM->LM              % compose 2 LMs sequentially
var
  d,d1: State lm,lm1,lm2:LM ml,ml1:ML n,m:Nat b:Bool
rew
  gt(LM0,lm)=F
  gt(seq1(d,lm),LM0)=T
  gt(seq1(d,lm),seq1(d1,lm1))
        =if(eq(lm,LM0),
           if(eq(lm1,LM0),gt(d,d1),F),
           if(eq(lm1,LM0),T,or(gt(d,d1),and(eq(d,d1),gt(lm,lm1)))))
  gt(seq1(d,lm),seqM(ml,lm1))=F
  gt(seqM(ml,lm),LM0)=T
  gt(seqM(ml,lm),seq1(d,lm1))=T
  gt(seqM(ml,lm),seqM(ml1,lm1))
        =if(eq(lm,LM0),
           if(eq(lm1,LM0),gt(ml,ml1),F),
           if(eq(lm1,LM0),T,or(gt(ml,ml1),and(eq(ml,ml1),gt(lm,lm1)))))
```

```
  conc(LM0,lm)=lm conc(lm,LM0)=lm
  conc(seq1(d,lm),lm1)=seq1(d,conc(lm,lm1))
  conc(seqM(ml,lm),lm1)=seqM(ml,conc(lm,lm1))

  conp(ML(lm),lm1)=conc(lm,lm1)
  conp(par(lm,ml),lm1)=seqM(par(lm,ml),lm1)

  eq(LM0,seq1(d,lm))=F eq(seq1(d,lm),LM0)=F
  eq(LM0,seqM(ml,lm))=F eq(seqM(ml,lm),LM0)=F
  eq(seq1(d,lm),seqM(ml,lm1))=F eq(seqM(ml,lm1),seq1(d,lm))=F

  eq(lm,lm)=T
  eq(seq1(d,lm),seq1(d1,lm1))=and(eq(d,d1),eq(lm,lm1))
  eq(seqM(ml,lm),seqM(ml1,lm1))=and(eq(ml,ml1),eq(lm,lm1))

  if(T,lm,lm1)=lm if(F,lm,lm1)=lm1 if(b,lm,lm)=lm if(not(b),lm,lm1)=if(b,lm1,lm)

  lenf(LM0)=0
  lenf(seq1(d,lm))=1
  lenf(seqM(ml,lm))=lenf(ml)

  % undefined getf1(LM0,n)=
  getf1(seq1(d,lm),0)=d
  getf1(seqM(ml,lm),n)=getf1(ml,n)

  replf1(seq1(d,lm),0,lm1)=conc(lm1,lm)
  replf1(seqM(ml,lm),n,lm1)=conp(replf1(ml,n,lm1),lm)

  remf1(lm,n)=replf1(lm,n,LM0)

  replf2(seqM(ml,lm),n,m,lm1,lm2)=conp(replf2(ml,n,m,lm1,lm2),lm)
  replremf2(lm,n,m,lm1)=replf2(lm,n,m,lm1,LM0)
  remf2(lm,n,m)=replf2(lm,n,m,LM0,LM0)

  seqc(lm,lm1)=conc(lm,lm1)
  parc(lm,lm1)=conp(comp(mkml(lm),mkml(lm1)),LM0)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort ML                                                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ML                          % Multiset of LM
func
  ML: LM->ML                     % Multiset with one list
  par: LM#ML->ML                 % Add a list to the multiset (first argument never LM0)
map
  eq: ML#ML->Bool                % equality on ML
  if:Bool#ML#ML->ML

  mkml :LM->ML                   % Make a proper ML out of an LM
  comp :ML#ML->ML                % Compose 2 MLs in a wf way.
  gt   :ML#ML->Bool

  in: LM#ML->Bool                % test if an lm is in ml (on the first level, of course).
  rem: LM#ML->ML                 % remove an lm from ml if it is on the first level,
                                 % don't change otherwise

  lenf: ML->Nat
  getf1: ML#Nat->State
  replf1: ML#Nat#LM->ML
  replf2:ML#Nat#Nat#LM#LM->ML    % replace n-th and m-th components
var
  d,d1: State lm,lm1,lm2:LM ml,ml1:ML n,m:Nat b:Bool
rew
  gt(ML(lm),ML(lm1))=gt(lm,lm1)
  gt(ML(lm),par(lm1,ml))=F
  gt(par(lm1,ml),ML(lm))=T
  gt(par(lm,ml),par(lm1,ml1))=or(gt(lm,lm1),and(eq(lm,lm1),gt(ml,ml1)))
```

```
mkml(LM0)=ML(LM0)
mkml(seq1(d,lm))=ML(seq1(d,lm))
mkml(seqM(ml,lm))=if(eq(lm,LM0),ml,ML(seqM(ml,lm)))

comp(ML(LM0),ml)=ml
comp(ml,ML(LM0))=ml

comp(ML(seq1(d,lm)),ML(seq1(d1,lm1)))=
      if(gt(seq1(d,lm),seq1(d1,lm1)),
        par(seq1(d1,lm1),ML(seq1(d,lm))),
        par(seq1(d,lm),ML(seq1(d1,lm1))))
comp(ML(seq1(d,lm)),ML(seqM(ml,lm1)))=par(seq1(d,lm1),ML(seqM(ml,lm1)))
comp(ML(seqM(ml,lm)),ML(seq1(d,lm1)))=comp(ML(seq1(d,lm1)),ML(seqM(ml,lm)))
comp(ML(seqM(ml,lm)),ML(seqM(ml1,lm1)))=
      if(gt(seqM(ml,lm),seqM(ml1,lm1)),
        par(seqM(ml1,lm1),ML(seqM(ml,lm))),
        par(seqM(ml,lm),ML(seqM(ml1,lm1))))

comp(ML(seq1(d,lm)),par(lm1,ml))=
      if(gt(seq1(d,lm),lm1),
        par(lm1,comp(ML(seq1(d,lm)),ml)),
        par(seq1(d,lm),par(lm1,ml)))
comp(par(lm1,ml),ML(seq1(d,lm)))=comp(ML(seq1(d,lm)),par(lm1,ml))
comp(ML(seqM(ml,lm)),par(lm1,ml1))=
      if(gt(seqM(ml,lm),lm1),
        par(lm1,comp(ML(seqM(ml,lm)),ml1)),
        par(seqM(ml,lm),par(lm1,ml1)))
comp(par(lm1,ml1),ML(seqM(ml,lm)))=comp(ML(seqM(ml,lm)),par(lm1,ml1))
comp(par(lm,ml),par(lm1,ml1))=
      if(gt(lm,lm1),
        par(lm1,comp(ml1,par(lm,ml))),
        par(lm,comp(ml,par(lm1,ml1))))

eq(ML(lm),par(lm1,ml))=F eq(par(lm1,ml),ML(lm))=F
eq(ML(lm1),ML(lm2))=eq(lm1,lm2)
eq(par(lm,ml),par(lm1,ml1))=                    % ML par(lm1,ml1) has at least 2 elements
      and(in(lm,par(lm1,ml1)),eq(ml,rem(lm,par(lm1,ml1))))
eq(ml,ml)=T

if(T,ml,ml1)=ml if(F,ml,ml1)=ml1 if(b,ml,ml)=ml if(not(b),ml,ml1)=if(b,ml1,ml)

in(lm,ML(lm1))=eq(lm,lm1)
in(lm,par(lm1,ml))=or(eq(lm,lm1),in(lm,ml))

% undefined (not needed) rem(lm,ML(lm1))=if(eq(lm,lm1),ML(LM0),ML(lm1))
rem(lm,par(lm1,ML(lm2)))=if(eq(lm,lm1),ML(lm2),if(eq(lm,lm2),ML(lm1),par(lm1,ML(lm2))))
rem(lm,par(lm1,par(lm2,ml)))=if(eq(lm,lm1),par(lm2,ml),par(lm1,rem(lm,par(lm2,ml))))

lenf(ML(lm))=lenf(lm)
lenf(par(lm,ml))=add(lenf(lm),lenf(ml))

getf1(ML(lm),n)=getf1(lm,n)
getf1(par(lm,ml),n)=if(gt(lenf(lm),n),getf1(lm,n),getf1(ml,sub(n,lenf(lm))))

replf1(ML(lm),n,lm1)=mkml(replf1(lm,n,lm1))
replf1(par(lm,ml),n,lm1)=if(gt(lenf(lm),n),
                           comp(mkml(replf1(lm,n,lm1)),ml),
                           comp(ML(lm),replf1(ml,sub(n,lenf(lm)),lm1)))

replf2(ML(lm),n,m,lm1,lm2)=mkml(replf2(lm,n,m,lm1,lm2))
replf2(par(lm,ml),n,m,lm1,lm2)=
      if(gt(m,n),
        if(gt(lenf(lm),n),
          if(gt(lenf(lm),m),
            comp(mkml(replf2(lm,n,m,lm1,lm2)),ml),
            comp(mkml(replf1(lm,n,lm1)),replf1(ml,sub(m,lenf(lm)),lm2))),
          comp(ML(lm),replf2(ml,sub(n,lenf(lm)),sub(m,lenf(lm)),lm1,lm2))),
        if(gt(lenf(lm),m),
          if(gt(lenf(lm),n),
```

```
                comp(mkml(replf2(lm,n,m,lm1,lm2)),ml),
                comp(mkml(replf1(lm,m,lm2)),replf1(ml,sub(n,lenf(lm)),lm1))),
              comp(ML(lm),replf2(ml,sub(n,lenf(lm)),sub(m,lenf(lm)),lm1,lm2))))

```

C.3 *ALM* and *AML* Data Types

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% ALM And AML data types                                         %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% sort Act (Actions)                                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort Act
func
   a:Nat->Act
map
  eq:Act#Act->Bool
  if:Bool#Act#Act->Act
  gt:Act#Act->Bool
var a,a1:Act n,m:Nat b:Bool
rew
  eq(a(n),a(m))=eq(n,m)
  if(T,a,a1)=a if(F,a,a1)=a1 if(b,a,a)=a if(not(b),a,a1)=if(b,a1,a)
  gt(a(n),a(m))=gt(n,m)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% sort ActSet (Sets of action Actions)                            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ActSet
func
  ActSet0:->ActSet
  _add:Act#ActSet->ActSet
map
  eq:ActSet#ActSet->Bool
  gt:ActSet#ActSet->Bool
  if:Bool#ActSet#ActSet->ActSet
  add:Act#ActSet->ActSet              % add an element
  add1:Act#ActSet->ActSet             % add an element assuming it is not in the set
  in:Act#ActSet->Bool                 % is an element in the set?
  rem:Act#ActSet->ActSet              % remove an element (if present)
  union:ActSet#ActSet->ActSet         % set union
  minus:ActSet#ActSet->ActSet         % set minus
  intersect:ActSet#ActSet->ActSet     % set intersection
var a,a1:Act as,as1:ActSet b:Bool
rew
  eq(as,as)=T eq(ActSet0,_add(a,as))=F eq(_add(a,as),ActSet0)=F
  eq(_add(a,as),_add(a1,as1))=and(in(a,_add(a1,as1)),eq(as,rem(a,_add(a1,as1))))

  gt(ActSet0,as)=F
  gt(_add(a,as),ActSet0)=T
  gt(_add(a,as),_add(a1,as1))=or(gt(a,a1),and(eq(a,a1),gt(as,as1)))

  if(T,as,as1)=as if(F,as,as1)=as1 if(b,as,as)=as if(not(b),as,as1)=if(b,as1,as)

  add(a,as)=if(in(a,as),as,add1(a,as))
  add1(a,ActSet0)=_add(a,ActSet0)
  add1(a,_add(a1,as))=if(gt(a,a1),_add(a1,add1(a,as)),_add(a,add(a1,as)))

  in(a,ActSet0)=F in(a,_add(a1,as))=or(eq(a,a1),in(a,as))

  rem(a,ActSet0)=ActSet0
  rem(a,_add(a1,as))=if(eq(a,a1),as,_add(a1,rem(a,as)))

  union(ActSet0,as)=as union(as,ActSet0)=as
```

```
  union(_add(a,as),as1)=union(as,add(a,as1))

  minus(ActSet0,as)=ActSet0 minus(as,ActSet0)=as
  minus(_add(a,as),as1)=if(in(a,as1),minus(as,as1),_add(a,minus(as,as1)))

  intersect(ActSet0,as)=ActSet0 intersect(as,ActSet0)=ActSet0
  intersect(_add(a,as),as1)=if(in(a,as1),_add(a,intersect(as,as1)),intersect(as,as1))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% sort ActMap (Function from Act to Act)                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ActMap
func
  ActMap0:->ActMap
  _add:Act#Act#ActMap->ActMap
map
  eq:ActMap#ActMap->Bool
  gt:ActMap#ActMap->Bool
  if:Bool#ActMap#ActMap->ActMap
  mod:Act#Act#ActMap->ActMap           % modify the mapping with the pair
  mod0:Act#Act#ActMap->ActMap          % modify the mapping with the pair assuming
                                       % the argument is there
  mod1:Act#Act#ActMap->ActMap          % modify the mapping with the pair assuming
                                       % the argument is not there
  in:Act#ActMap->Bool                  % is an element in the map's args?
  in:Act#Act#ActMap->Bool              % is a pair in the map?
  rem:Act#ActMap->ActMap               % remove a pair by the arg (if present)
  appl:Act#ActMap->Act                 % apply the mapping
  comp:ActMap#ActMap->ActMap           % compose 2 maps
  rimage:ActSet#ActMap->ActSet         % F^{-1}(AS)
  simpl:ActSet#ActMap->ActMap          % transform am not to change as
var a,a1,a2,a3:Act as:ActSet am,am1:ActMap b:Bool
rew
  eq(am,am)=T eq(ActMap0,_add(a,a1,am))=F eq(_add(a,a1,am),ActMap0)=F
  eq(_add(a,a1,am),_add(a2,a3,am1))=and(in(a,a1,_add(a1,a2,am1)),eq(am,rem(a,_add(a2,a3,am1))))

  gt(ActMap0,am)=F
  gt(_add(a,a1,am),ActMap0)=T
  gt(_add(a,a1,am),_add(a2,a3,am1))=
        or(gt(a2,a),and(eq(a2,a),or(gt(a1,a3),and(eq(a1,a3),gt(am,am1)))))

  if(T,am,am1)=am if(F,am,am1)=am1 if(b,am,am)=am if(not(b),am,am1)=if(b,am1,am)

  mod(a,a1,am)=if(in(a,am),mod0(a,a1,am),mod1(a,a1,am))

  mod0(a,a1,_add(a2,a3,am))=if(eq(a,a2),_add(a2,a1,am),_add(a2,a3,mod0(a,a1,am)))
  mod1(a,a1,ActMap0)=_add(a,a1,ActMap0)
  mod1(a,a1,_add(a2,a3,am))=if(gt(a,a2),_add(a2,a3,mod1(a,a1,am)),_add(a,a1,_add(a2,a3,am)))

  in(a,ActMap0)=F in(a,_add(a2,a3,am))=or(eq(a,a2),in(a,am))
  in(a,a1,ActMap0)=F in(a,a1,_add(a2,a3,am))=or(and(eq(a,a2),eq(a1,a3)),in(a,a1,am))

  rem(a,ActMap0)=ActMap0
  rem(a,_add(a2,a3,am))=if(eq(a,a2),am,_add(a2,a3,rem(a,am)))

  appl(a,ActMap0)=a appl(a,_add(a2,a3,am))=if(eq(a,a2),a3,appl(a,am))

  comp(ActMap0,am)=am comp(am,ActMap0)=am
  comp(_add(a,a1,am),am1)=
        if(eq(appl(a1,am1),a),rem(a,comp(am,am1)),mod1(a,appl(a1,am1),rem(a,comp(am,am1))))

  rimage(ActSet0,am)=ActSet0 rimage(as,ActMap0)=as
  rimage(as,_add(a,a1,am))=if(in(a1,as),add(a,rimage(as,am)),rem(a,rimage(as,am)))

  simpl(ActSet0,am)=am simpl(as,ActMap0)=ActMap0
  simpl(as,_add(a,a1,am))=if(in(a,as),simpl(as,am),_add(a,a1,simpl(as,am)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% sort Annote (Triple of one ActMap and two ActSets)        %%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort Annote
func
  ann:ActMap#ActSet#ActSet->Annote
map
  eq:Annote#Annote->Bool
  gt:Annote#Annote->Bool
  if:Bool#Annote#Annote->Annote
  Ann0:->Annote
  comp:Annote#Annote->Annote
  getH:Annote->ActSet
  getI:Annote->ActSet
  getR:Annote->ActMap
var as,as1,as2,as3:ActSet am,am1:ActMap ann1,ann2:Annote b:Bool
rew
  eq(ann(am,as,as1),ann(am1,as2,as3))=and(and(eq(am,am1),eq(as,as2)),eq(as1,as3))

  gt(ann(am,as,as1),ann(am1,as2,as3))=
        or(gt(as1,as3),and(eq(as1,as3),or(gt(as,as2),and(eq(as,as2),gt(am,am1)))))

  if(T,ann1,ann2)=ann1 if(F,ann1,ann2)=ann2
  if(b,ann1,ann1)=ann1 if(not(b),ann1,ann2)=if(b,ann2,ann1)

  Ann0=ann(ActMap0,ActSet0,ActSet0)

  comp(ann(am,as,as1),ann(am1,as2,as3))=
        ann(comp(am1,am),union(as2,rimage(as,am1)),union(as3,minus(rimage(as1,am1),as2)))

  getH(ann(am,as,as1))=as1 getI(ann(am,as,as1))=as getR(ann(am,as,as1))=am

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  ALM                                                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ALM                                   % List of AML or State elements
func
  ALM0: ->ALM                              % Empty list
  seq1: Annote#State#ALM->ALM              % Add one State element to the head of the list
  seqM: AML#ALM->ALM                       % Add one AML to the head of the list
                                           % (first argument never AML(x))
map
  eq: ALM#ALM->Bool                        % Equality on ALM
  if: Bool#ALM#ALM->ALM

  gt: ALM#ALM->Bool
  conc: ALM#ALM->ALM                       % Concatenate 2 ALMs in a wf way.
  conp: AML#ALM->ALM                       % Prepend an AML to an ALM
  annote: Annote#ALM->ALM                  % add annotation

  lenf:ALM->Nat                            % number of "ready" components
  getf1d:ALM#Nat->State                    % get n-th component
  getf1a:ALM#Nat->Annote                   % get n-th component's annotation
  replf1:ALM#Nat#ALM->ALM                  % replace n-th component
  remf1:ALM#Nat->ALM                       % remove n-th component

  getf2a0:ALM#Nat#Nat->Annote              % get n-th component's annotation
  getf2a1:ALM#Nat#Nat->Annote              % get m-th component's annotation
  getf2a:ALM#Nat#Nat->Annote               % get (n,m)-th components' annotation

  replf2:ALM#Nat#Nat#ALM#ALM->ALM          % replace n-th and m-th components
  replremf2:ALM#Nat#Nat#ALM->ALM           % replace n-th and remove m-th components
  remf2:ALM#Nat#Nat->ALM                   % remove n-th and m-th components

  parc: Annote#ALM#ALM->ALM                % Compose 2 ALMs parallely
  seqc: ALM#ALM->ALM                       % Compose 2 ALMs sequentially
var
  d,d1: State lm,lm1,lm2:ALM ml,ml1:AML n,m:Nat b:Bool ann,ann1:Annote a:Act
rew
  gt(ALM0,lm)=F
  gt(seq1(ann,d,lm),ALM0)=T
```

```
gt(seq1(ann,d,lm),seq1(ann1,d1,lm1))
      =if(eq(eq(lm,ALM0),eq(lm1,ALM0)),
        if(eq(eq(ann,Ann0),eq(ann1,Ann0)),
          or(gt(d,d1),and(eq(d,d1),or(gt(lm,lm1),and(eq(lm,lm1),gt(ann,ann1))))),
          eq(ann1,Ann0)),
        eq(lm1,ALM0))
gt(seq1(ann,d,lm),seqM(ml,lm1))=F
gt(seqM(ml,lm),ALM0)=T
gt(seqM(ml,lm),seq1(ann,d,lm1))=T
gt(seqM(ml,lm),seqM(ml1,lm1))
      =if(eq(eq(lm,ALM0),eq(lm1,ALM0)),
        or(gt(ml,ml1),and(eq(ml,ml1),gt(lm,lm1))),
        eq(lm1,ALM0))

conc(ALM0,lm)=lm conc(lm,ALM0)=lm
conc(seq1(ann,d,lm),lm1)=seq1(ann,d,conc(lm,lm1))
conc(seqM(ml,lm),lm1)=seqM(ml,conc(lm,lm1))

conp(AML(lm),lm1)=conc(lm,lm1)
conp(par(ann,lm,ml),lm1)=seqM(par(ann,lm,ml),lm1)

annote(ann,ALM0)=ALM0 annote(Ann0,lm)=lm
annote(ann,seq1(ann1,d,lm))=seq1(comp(ann,ann1),d,annote(ann,lm))
annote(ann,seqM(ml,lm))=conp(annote(ann,ml),annote(ann,lm))

eq(ALM0,seq1(ann,d,lm))=F eq(seq1(ann,d,lm),ALM0)=F
eq(ALM0,seqM(ml,lm))=F eq(seqM(ml,lm),ALM0)=F
eq(seq1(ann,d,lm),seqM(ml,lm1))=F eq(seqM(ml,lm1),seq1(ann,d,lm))=F

eq(lm,lm)=T
eq(seq1(ann,d,lm),seq1(ann1,d1,lm1))=and(and(eq(d,d1),eq(lm,lm1)),eq(ann,ann1))
eq(seqM(ml,lm),seqM(ml1,lm1))=and(eq(ml,ml1),eq(lm,lm1))

if(T,lm,lm1)=lm if(F,lm,lm1)=lm1 if(b,lm,lm)=lm if(not(b),lm,lm1)=if(b,lm1,lm)

lenf(ALM0)=0
lenf(seq1(ann,d,lm))=1
lenf(seqM(ml,lm))=lenf(ml)

% undefined getf1(ALM0,n)=
getf1d(seq1(ann,d,lm),0)=d
getf1d(seqM(ml,lm),n)=getf1d(ml,n)
getf1a(seq1(ann,d,lm),0)=ann
getf1a(seqM(ml,lm),n)=getf1a(ml,n)

replf1(seq1(ann,d,lm),0,lm1)=conc(annote(ann,lm1),lm)
replf1(seqM(ml,lm),n,lm1)=conp(replf1(ml,n,lm1),lm)

remf1(lm,n)=replf1(lm,n,ALM0)

getf2a0(seqM(ml,lm),n,m)=getf2a0(ml,n,m)
getf2a1(seqM(ml,lm),n,m)=getf2a1(ml,n,m)
getf2a(seqM(ml,lm),n,m)=getf2a(ml,n,m)

replf2(seqM(ml,lm),n,m,lm1,lm2)=conp(replf2(ml,n,m,lm1,lm2),lm)
replremf2(lm,n,m,lm1)=replf2(lm,n,m,lm1,ALM0)
remf2(lm,n,m)=replf2(lm,n,m,ALM0,ALM0)

seqc(lm,lm1)=conc(lm,lm1)
parc(ann,lm,lm1)=annote(ann,conp(comp(mkml(lm),mkml(lm1)),ALM0))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort AML                                                               %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort AML                               % Multiset of ALM
func
  AML: ALM->AML                        % Multiset with one list
  par: Annote#ALM#AML->AML             % Add a list to the multiset (first argument never ALM0)
map
```

```
  eq: AML#AML->Bool                     % equality on AML
  if:Bool#AML#AML->AML

  mkml :ALM->AML                        % Make a proper AML out of an ALM
  comp :AML#AML->AML                    % Compose 2 AMLs in a wf way.
  annote: Annote#AML->AML               % add annotation
  gt   :AML#AML->Bool

  in: ALM#AML->Bool                     % test if an lm is in ml (on the first level)
  rem: ALM#AML->AML                     % remove an lm from ml if it is on the first level,
                                        % don't change otherwise

  lenf: AML->Nat
  getf1d: AML#Nat->State
  getf1a: AML#Nat->Annote
  replf1: AML#Nat#ALM->AML

  getf2a0:AML#Nat#Nat->Annote           % get n-th component's annotation
  getf2a1:AML#Nat#Nat->Annote           % get m-th component's annotation
  getf2a:AML#Nat#Nat->Annote            % get (n,m)-th components' annotation
  replf2:AML#Nat#Nat#ALM#ALM->AML       % replace n-th and m-th components
var
  d,d1: State lm,lm1,lm2:ALM ml,ml1:AML n,m:Nat b:Bool ann,ann1,ann2:Annote a:Act
rew
  gt(AML(lm),AML(lm1))=gt(lm,lm1)
  gt(AML(lm),par(ann,lm1,ml))=F
  gt(par(ann,lm1,ml),AML(lm))=T
  gt(par(ann,lm,ml),par(ann1,lm1,ml1))
        =if(eq(eq(ann,Ann0),eq(ann1,Ann0)),
          or(gt(lm,lm1),and(eq(lm,lm1),gt(ml,ml1))),
          eq(ann1,Ann0))

  mkml(ALM0)=AML(ALM0)
  mkml(seq1(ann,d,lm))=AML(seq1(ann,d,lm))
  mkml(seqM(ml,lm))=if(eq(lm,ALM0),ml,AML(seqM(ml,lm)))

  comp(AML(ALM0),ml)=ml
  comp(ml,AML(ALM0))=ml

  comp(AML(seq1(ann,d,lm)),AML(seq1(ann1,d1,lm1)))=
        if(gt(seq1(ann,d,lm),seq1(ann1,d1,lm1)),
          par(Ann0,seq1(ann1,d1,lm1),AML(seq1(ann1,d,lm))),
          par(Ann0,seq1(ann,d,lm),AML(seq1(ann1,d1,lm1))))
  comp(AML(seq1(ann,d,lm)),AML(seqM(ml,lm1)))=par(Ann0,seq1(ann,d,lm1),AML(seqM(ml,lm1)))
  comp(AML(seqM(ml,lm)),AML(seq1(ann,d,lm1)))=comp(AML(seq1(ann,d,lm1)),AML(seqM(ml,lm)))
  comp(AML(seqM(ml,lm)),AML(seqM(ml1,lm1)))=
        if(gt(seqM(ml,lm),seqM(ml1,lm1)),
          par(Ann0,seqM(ml1,lm1),AML(seqM(ml,lm))),
          par(Ann0,seqM(ml,lm),AML(seqM(ml1,lm1))))

  comp(AML(seq1(ann,d,lm)),par(ann1,lm1,ml))=
        if(and(eq(ann1,Ann0),gt(seq1(ann,d,lm),lm1)),
          par(Ann0,lm1,comp(AML(seq1(ann,d,lm)),ml)),
          par(Ann0,seq1(ann,d,lm),par(ann1,lm1,ml)))
  comp(par(ann1,lm1,ml),AML(seq1(ann,d,lm)))=comp(AML(seq1(ann,d,lm)),par(ann1,lm1,ml))
  comp(AML(seqM(ml,lm)),par(ann1,lm1,ml1))=
        if(and(eq(ann1,Ann0),gt(seqM(ml,lm),lm1)),
          par(Ann0,lm1,comp(AML(seqM(ml,lm)),ml1)),
          par(Ann0,seqM(ml,lm),par(ann1,lm1,ml1)))
  comp(par(ann1,lm1,ml1),AML(seqM(ml,lm)))=comp(AML(seqM(ml,lm)),par(ann1,lm1,ml1))
  comp(par(ann,lm,ml),par(ann1,lm1,ml1))=
        if(eq(ann,Ann0),
          if(eq(ann1,Ann0),
            if(gt(lm,lm1),
              par(Ann0,lm1,comp(ml1,par(ann,lm,ml))),
              par(Ann0,lm,comp(ml,par(ann1,lm1,ml1)))),
            par(Ann0,lm,comp(ml,par(ann1,lm1,ml1)))),
          if(eq(ann1,Ann0),
            par(Ann0,lm1,comp(ml1,par(ann,lm,ml))),
```

```
          if(gt(par(ann,lm,ml),par(ann1,lm1,ml1)),
            par(Ann0,seqM(par(ann1,lm1,ml1),ALM0),par(ann,lm,ml)),
            par(Ann0,seqM(par(ann,lm,ml),ALM0),par(ann1,lm1,ml1)))))

annote(ann,AML(lm))=mkml(annote(ann,lm))
annote(ann,par(ann1,lm,ml))=par(comp(ann,ann1),lm,ml)

eq(AML(lm),par(ann1,lm1,ml))=F eq(par(ann1,lm1,ml),AML(lm))=F
eq(AML(lm1),AML(lm2))=eq(lm1,lm2)
eq(par(ann,lm,ml),par(ann1,lm1,ml1))=
      and(eq(ann,ann1),and(in(lm,par(Ann0,lm1,ml1)),eq(ml,rem(lm,par(Ann0,lm1,ml1)))))
                                               % AML par(Ann0,lm1,ml1) has at least 2 elements
eq(ml,ml)=T

if(T,ml,ml1)=ml if(F,ml,ml1)=ml1 if(b,ml,ml)=ml if(not(b),ml,ml1)=if(b,ml1,ml)

in(lm,AML(lm1))=eq(lm,lm1)
in(lm,par(ann1,lm1,ml))=and(eq(ann1,Ann0),or(eq(lm,lm1),in(lm,ml)))

% undefined (not needed) rem(lm,AML(lm1))=if(eq(lm,lm1),AML(ALM0),AML(lm1))
rem(lm,par(ann1,lm1,AML(lm2)))=
      if(eq(ann1,Ann0),
        if(eq(lm,lm1),AML(lm2),if(eq(lm,lm2),AML(lm1),par(ann1,lm1,AML(lm2)))),
        par(ann1,lm1,AML(lm2)))
rem(lm,par(ann1,lm1,par(ann2,lm2,ml)))=
      if(eq(ann1,Ann0),
        if(eq(lm,lm1),par(ann2,lm2,ml),par(ann1,lm1,rem(lm,par(ann2,lm2,ml)))),
        par(ann1,lm1,par(ann2,lm2,ml)))

lenf(AML(lm))=lenf(lm)
lenf(par(ann,lm,ml))=add(lenf(lm),lenf(ml))

getf1d(AML(lm),n)=getf1d(lm,n)
getf1d(par(ann,lm,ml),n)=if(gt(lenf(lm),n),getf1d(lm,n),getf1d(ml,sub(n,lenf(lm))))
getf1a(AML(lm),n)=getf1a(lm,n)
getf1a(par(ann,lm,ml),n)=comp(ann,if(gt(lenf(lm),n),getf1a(lm,n),getf1a(ml,sub(n,lenf(lm)))))

replf1(AML(lm),n,lm1)=mkml(replf1(lm,n,lm1))
replf1(par(ann,lm,ml),n,lm1)=annote(ann,if(gt(lenf(lm),n),
                            comp(mkml(replf1(lm,n,lm1)),ml),
                            comp(AML(lm),replf1(ml,sub(n,lenf(lm)),lm1))))

getf2a0(AML(lm),n,m)=getf2a0(lm,n,m)
getf2a0(par(ann,lm,ml),n,m)=
      if(eq(gt(lenf(lm),n),gt(lenf(lm),m)),
        if(gt(lenf(lm),n),getf2a0(lm,n,m),getf2a0(ml,sub(n,lenf(lm)),sub(m,lenf(lm)))),
        if(gt(lenf(lm),n),getf1a(lm,n),getf1a(ml,sub(n,lenf(lm)))))
getf2a1(AML(lm),n,m)=getf2a1(lm,n,m)
getf2a1(par(ann,lm,ml),n,m)=
      if(eq(gt(lenf(lm),n),gt(lenf(lm),m)),
        if(gt(lenf(lm),n),getf2a1(lm,n,m),getf2a1(ml,sub(n,lenf(lm)),sub(m,lenf(lm)))),
        if(gt(lenf(lm),m),getf1a(lm,n),getf1a(ml,sub(m,lenf(lm)))))
getf2a(AML(lm),n,m)=getf2a(lm,n,m)
getf2a(par(ann,lm,ml),n,m)=
      if(eq(gt(lenf(lm),n),gt(lenf(lm),m)),
        comp(ann,
              if(gt(lenf(lm),n),
                getf2a(lm,n,m),
                getf2a(ml,sub(n,lenf(lm)),sub(m,lenf(lm))))),
        ann)

replf2(AML(lm),n,m,lm1,lm2)=mkml(replf2(lm,n,m,lm1,lm2))
replf2(par(ann,lm,ml),n,m,lm1,lm2)=annote(ann,
      if(gt(m,n),
        if(gt(lenf(lm),n),
          if(gt(lenf(lm),m),
            comp(mkml(replf2(lm,n,m,lm1,lm2)),ml),
            comp(mkml(replf1(lm,n,lm1)),replf1(ml,sub(m,lenf(lm)),lm2))),
          comp(AML(lm),replf2(ml,sub(n,lenf(lm)),sub(m,lenf(lm)),lm1,lm2))),
```

```
          if(gt(lenf(lm),m),
            if(gt(lenf(lm),n),
              comp(mkml(replf2(lm,n,m,lm1,lm2)),ml),
              comp(mkml(replf1(lm,m,lm2)),replf1(ml,sub(n,lenf(lm)),lm1))),
            comp(AML(lm),replf2(ml,sub(n,lenf(lm)),sub(m,lenf(lm)),lm1,lm2)))))
```

C.4 Basic Data Types for Multi-Party Communications

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  Sorts  LNat, LState; ActPars, E_i, LActPars, LE_i, functions on them (all generated)  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  LNat (list of Naturals)                             %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LNat
func
  LNat0:->LNat
  add:Nat#LNat->LNat
map
  eq:LNat#LNat->Bool
  if:Bool#LNat#LNat->LNat
  len:LNat->Nat
  cat:LNat#LNat->LNat
  head:LNat->Nat                 % return the head of the list
  in:Nat#LNat->Bool
  lower:LNat#Nat->LNat           % return a sublist containing elems <n
  upper:LNat#Nat->LNat           % return a sublist containing elems >=n
  sub:LNat#Nat->LNat             % subtract n from each elem.
  is_unique:LNat->Bool           % are all the elems different?
  is_sorted:LNat->Bool           % is the list sorted?
  is_each_lower:Nat#LNat->Bool   % is each of the elems lower than the first arg?
  gen0Mm1:Nat->LNat              % generate list 0..M-1 (if M=0 then return LNat0)
var
  lnat,lnat1:LNat n,m:Nat b:Bool
rew
  eq(lnat,lnat)=T eq(LNat0,add(n,lnat))=F
  eq(add(n,lnat),LNat0)=F eq(add(n,lnat),add(m,lnat1))=and(eq(n,m),eq(lnat,lnat1))
  if(T,lnat,lnat1)=lnat if(F,lnat,lnat1)=lnat1
  if(b,lnat,lnat)=lnat if(not(b),lnat,lnat1)=if(b,lnat1,lnat)
  len(LNat0)=0
  len(add(n,lnat))=succ(len(lnat))
  cat(LNat0,lnat)=lnat cat(lnat,LNat0)=lnat
  cat(add(n,lnat),lnat1)=add(n,cat(lnat,lnat1))
  head(add(n,lnat))=n
  in(n,LNat0)=F
  in(n,add(m,lnat))=or(eq(n,m),in(n,lnat))
  lower(LNat0,n)=LNat0
  lower(add(m,lnat),n)=if(gt(n,m),add(m,lower(lnat,n)),lower(lnat,n))
  upper(LNat0,n)=LNat0
  upper(add(m,lnat),n)=if(gt(n,m),upper(lnat,n),add(m,upper(lnat,n)))
  sub(LNat0,n)=LNat0
  sub(add(m,lnat),n)=add(sub(m,n),sub(lnat,n))
  is_unique(LNat0)=T
  is_unique(add(n,lnat))=and(in(n,lnat),is_unique(lnat))
  is_sorted(LNat0)=T is_sorted(add(n,LNat0))=T
  is_sorted(add(n,add(m,lnat)))=and(not(gt(n,m)),is_sorted(add(m,lnat)))
  is_each_lower(n,LNat0)=T
  is_each_lower(n,add(m,lnat))=and(gt(n,m),is_each_lower(n,lnat))
  gen0Mm1(0)=LNat0
  gen0Mm1(x2p1(n))=cat(gen0Nm1(x2p0(n)),add(n,LNat0))
  gen0Mm1(x2p2(n))=cat(gen0Nm1(x2p1(n)),add(n,LNat0))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%  sort  LState  (list of States)                                 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LState
func
  LState0:->LState
  add:State#LState->LState
map
  eq:LState#LState->Bool
  if:Bool#LState#LState->LState
  len:LState->Nat
var
  ld,ld1:LState d,d1:State b:Bool
rew
  eq(ld,ld)=T eq(LState0,add(d,ld))=F
  eq(add(d,ld),LState0)=F eq(add(d,ld),add(d1,ld1))=and(eq(d,d1),eq(ld,ld1))
  if(T,ld,ld1)=ld if(F,ld,ld1)=ld1 if(b,ld,ld)=ld if(not(b),ld,ld1)=if(b,ld1,ld)
  len(LState0)=0
  len(add(d,ld))=succ(len(ld))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Sorts LActPars (list of ActPars, defined below)            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LActPars
func
  LActPars0:->LActPars
  add:ActPars#LActPars->LActPars
map
  eq:LActPars#LActPars->Bool
  if:Bool#LActPars#LActPars->LActPars
  len:LActPars->Nat
  head:LActPars->ActPars        % return the head of the list
  EQ:LActPars->Bool             % are all of the elements equal?
var
  laa,laa1:LActPars aa,aa1:ActPars b:Bool
rew
  eq(laa,laa)=T eq(LActPars0,add(aa,laa))=F
  eq(add(aa,laa),LActPars0)=F eq(add(aa,laa),add(aa1,laa1))=and(eq(aa,aa1),eq(laa,laa1))
  if(T,laa,laa1)=laa if(F,laa,laa1)=laa1 if(b,laa,laa)=laa if(not(b),laa,laa1)=if(b,laa1,laa)
  len(LActPars0)=0
  len(add(aa,laa))=succ(len(laa))
  head(add(aa,laa))=aa
  EQ(LActPars0)=T
  EQ(add(aa,LActPars0))=T
  EQ(add(aa,add(aa1,laa)))=and(eq(aa,aa1),EQ(add(aa1,laa)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  To be generated from the spec                              %%%
%%%%  The parts that do not parse before actual generation       %%%
%%%%  are commented out                                          %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Sort ActPars (unique action parameters tuples)                     %%%
%%% if parameters of act(m) are a_k(...),                               %%%
%%% it means that act(m) and act(k) have the same parameter sorts       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ActPars
func
  % a_k:D_0#...#D_n->ActPars
map
  eq: ActPars#ActPars->Bool
  gt: ActPars#ActPars->Bool
  if: Bool#ActPars#ActPars->ActPars
  % pr_k_0:ActPars->D_0 ... pr_k_n:ActPars->D_n
var
  aa,aa1:ActPars b:Bool
rew
  if(T,aa,aa1)=aa if(F,aa,aa1)=aa1 if(b,aa,aa)=aa if(not(b),aa,aa1)=if(b,aa1,aa)
  % gt(a_k(d0,...,dn),a_k(e0,...,en))=
```

```
  %     or(gt(d0,e0),and(eq(d0,e0),...
  %             or(gt(d{n-1},e{n-1}),and(eq(d{n-1},e{n-1}),gt(dn,en)))...))
  % gt(a_k(d0,...,dn),a_m(e0,...,el))="k>m"
  eq(aa,aa)=T
  % eq(a_k(d0,...,dn),a_k(e0,...,en))=and(eq(d0,e0),...,and(eq(dn,en))...)
  % eq(a_k(d0,...,dn),a_m(e0,...,el))=F    (k!=m)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Sorts E_i (sum types tuples)                                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort E_0
func
  % e_i:D_0#...#D_n->E_i
map
  eq: E_0#E_0->Bool
  gt: E_0#E_0->Bool
  if: Bool#E_0#E_0->E_0
  % pr_0:E_i->D_0 ... pr_n:E_i->D_n
var
  ee,ee1:E_0 b:Bool
rew
  if(T,ee,ee1)=ee if(F,ee,ee1)=ee1 if(b,ee,ee)=ee if(not(b),ee,ee1)=if(b,ee1,ee)
  % gt(e_i(d0,...,dn),e_i(e0,...,en))=
  %     or(gt(d0,e0),and(eq(d0,e0),...
  %             or(gt(d{n-1},e{n-1}),and(eq(d{n-1},e{n-1}),gt(dn,en)))...))
  eq(ee,ee)=T
  % eq(e_i(d0,...,dn),e_i(e0,...,en))=and(eq(d0,e0),...,and(eq(dn,en))...)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Sorts LE_i (list of E_i)                                       %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LE_0
func
  LE0_0:->LE_0
  add:E_0#LE_0->LE_0
map
  eq:LE_0#LE_0->Bool
  if:Bool#LE_0#LE_0->LE_0
  len:LE_0->Nat
  head:LE_0->E_0
var
  lee,lee1:LE_0 ee,ee1:E_0 b:Bool
rew
  eq(lee,lee)=T eq(LE0_0,add(ee,lee))=F
  eq(add(ee,lee),LE0_0)=F eq(add(ee,lee),add(ee1,lee1))=and(eq(ee,ee1),eq(lee,lee1))
  if(T,lee,lee1)=lee if(F,lee,lee1)=lee1 if(b,lee,lee)=lee if(not(b),lee,lee1)=if(b,lee1,lee)
  len(LE0_0)=0  len(add(ee,lee))=succ(len(lee))
  head(add(ee,lee))=ee

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Functions F_i and C_i (use the terms vectors f_i and c_i)  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  F_0:LState#LE_0->LActPars
  C_0:LState#LE_0->Bool
var
  d:State ld:LState e:E_0 le:LE_0
rew
  F_0(LState0,LE0_0)=LActPars0
  % F_i(add(d,ld),add(e,le))=add([meta(f_i)](pr_k(d),pr_k(e)),F_i(ld,le))
  C_0(LState0,LE0_0)=T
  % C_i(add(d,ld),add(e,le))=and([meta(c_i)](pr_k(d),pr_k(e)),C_i(ld,le))
```

C.5 Data Types for Multi-Party Communications with *LM* and *ML*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  LLM  (list of LMs)                                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LLM
func
  LLM0:->LLM
  add:LM#LLM->LLM
map
  eq:LLM#LLM->Bool
  if:Bool#LLM#LLM->LLM
  len:LLM->Nat
  cat:LLM#LLM->LLM
  lower:LLM#LNat#Nat->LLM       % return a sublist containing elems whose places are <n
  upper:LLM#LNat#Nat->LLM       % return a sublist containing elems whose places are >=n
  LEmptyLM:Nat->LLM             % returns the list consisting of n LM0s.
var
  llm,llm1:LLM lnat:LNat lm,lm1:LM b:Bool n,m:Nat
rew
  eq(llm,llm)=T eq(LLM0,add(lm,llm))=F
  eq(add(lm,llm),LLM0)=F eq(add(lm,llm),add(lm1,llm1))=and(eq(lm,lm1),eq(llm,llm1))
  if(T,llm,llm1)=llm if(F,llm,llm1)=llm1 if(b,llm,llm)=llm if(not(b),llm,llm1)=if(b,llm1,llm)
  len(LLM0)=0  len(add(lm,llm))=succ(len(llm))
  cat(LLM0,llm)=llm cat(llm,LLM0)=llm  cat(add(lm,llm),llm1)=add(lm,cat(llm,llm1))
  lower(LLM0,LNat0,n)=LLM0
  lower(add(lm,llm),add(m,lnat),n)=if(gt(n,m),add(lm,lower(llm,lnat,n)),lower(llm,lnat,n))
  upper(LLM0,LNat0,n)=LLM0
  upper(add(lm,llm),add(m,lnat),n)=if(gt(n,m),upper(llm,lnat,n),add(lm,upper(llm,lnat,n)))
  LEmptyLM(0)=LLM0
  LEmptyLM(x2p1(n))=add(LM0,cat(LEmptyLM(n),LEmptyLM(n)))
  LEmptyLM(x2p2(n))=add(LM0,LEmptyLM(x2p1(n)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Additional parts for the sorts LM and ML                               %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  getfn:LM#LNat->LState          % get the list of states.
  replfn:LM#LNat#LLM->LM         % replace the components with indices from
                                 % LNat with the elements of LLM
  replfn:ML#LNat#LLM->ML
  remfn:LM#LNat->LM              % remove the components with indices from LNat
var
  llm:LLM lnat:LNat lm,lm1:LM ml:ML n:Nat
rew
  getfn(lm,LNat0)=LState0
  getfn(lm,add(n,lnat))=add(getf1(lm,n),getfn(lm,lnat))

  replfn(lm,add(n,LNat0),add(lm1,LLM0))=replf1(lm,n,lm1)
  replfn(seqM(ml,lm),add(n,lnat),add(lm1,llm))=conp(replfn(ml,add(n,lnat),add(lm1,llm)),lm)

  replfn(ML(lm),lnat,llm)=mkml(replfn(lm,lnat,llm))
  replfn(par(lm,ml),lnat,llm)=
    comp(if(eq(lower(lnat,lenf(lm)),LNat0),
           ML(LM0),
           mkml(replfn(lm,lower(lnat,lenf(lm)),lower(llm,lnat,lenf(lm))))),
         if(eq(upper(lnat,lenf(lm)),LNat0),
           ML(LM0),
           replfn(ml,sub(upper(lnat,lenf(lm)),lenf(lm)),upper(llm,lnat,lenf(lm)))))

  remfn(lm,lnat)=replfn(lm,lnat,LEmptyLM(len(lnat)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  To be generated from the spec                                         %%%
%%%%  The parts that do not parse before actual generation                  %%%
%%%%  are commented out                                                     %%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Functions mkllm_i                                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  mkllm_0:LState#LE_0->LLM
var
  d:State ld:LState ee:E_0 lee:LE_0
rew
  mkllm_0(LState0,LE0_0)=add(LM0,LLM0)
  % mkllm_0(add(d,ld),add(ee,lee))=add([meta(mklm_0)](pr_k(d),pr_k(ee)),mkllm_0(ld,lee))
```

C.6 Data Types for Multi-Party Communications with *ALM* and *AML*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  LALM  (list of ALMs)                                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LALM
func
  LALM0:->LALM
  add:ALM#LALM->LALM
map
  eq:LALM#LALM->Bool
  if:Bool#LALM#LALM->LALM
  len:LALM->Nat
  cat:LALM#LALM->LALM
  lower:LALM#LNat#Nat->LALM      % return a sublist containing elems whose places are <n
  upper:LALM#LNat#Nat->LALM      % return a sublist containing elems whose places are >=n
  LEmptyALM:Nat->LALM            % returns the list consisting of n ALM0s.
var
  llm,llm1:LALM lnat:LNat lm,lm1:ALM b:Bool n,m:Nat
rew
  eq(llm,llm)=T eq(LALM0,add(lm,llm))=F
  eq(add(lm,llm),LALM0)=F eq(add(lm,llm),add(lm1,llm1))=and(eq(lm,lm1),eq(llm,llm1))
  if(T,llm,llm1)=llm if(F,llm,llm1)=llm1 if(b,llm,llm)=llm if(not(b),llm,llm1)=if(b,llm1,llm)
  len(LALM0)=0  len(add(lm,llm))=succ(len(llm))
  cat(LALM0,llm)=llm  cat(llm,LALM0)=llm  cat(add(lm,llm),llm1)=add(lm,cat(llm,llm1))
  lower(LALM0,LNat0,n)=LALM0
  lower(add(lm,llm),add(m,lnat),n)=if(gt(n,m),add(lm,lower(llm,lnat,n)),lower(llm,lnat,n))
  upper(LALM0,LNat0,n)=LALM0
  upper(add(lm,llm),add(m,lnat),n)=if(gt(n,m),upper(llm,lnat,n),add(lm,upper(llm,lnat,n)))
  LEmptyALM(0)=LALM0
  LEmptyALM(x2p1(n))=add(ALM0,cat(LEmptyALM(n),LEmptyALM(n)))
  LEmptyALM(x2p2(n))=add(ALM0,LEmptyALM(x2p1(n)))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  sort  LAct  (list of Actions)                                  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LAct
func
  LAct0:->LAct
  add:Act#LAct->LAct
map
  eq:LAct#LAct->Bool
  if:Bool#LAct#LAct->LAct
  len:LAct->Nat
  cat:LAct#LAct->LAct
  lower:LAct#LNat#Nat->LAct      % return a sublist containing elems whose places are <n
  upper:LAct#LNat#Nat->LAct      % return a sublist containing elems whose places are >=n
  mklact:Nat#Act->LAct           % generate the list of n actions a
var
  la,la1:LAct lnat:LNat a,a1:Act b:Bool n,m:Nat
rew
```

```
  eq(la,la)=T eq(LAct0,add(a,la))=F
  eq(add(a,la),LAct0)=F eq(add(a,la),add(a1,la1))=and(eq(a,a1),eq(la,la1))
  if(T,la,la1)=la if(F,la,la1)=la1 if(b,la,la)=la if(not(b),la,la1)=if(b,la1,la)
  len(LAct0)=0  len(add(a,la))=succ(len(la))
  cat(LAct0,la)=la cat(la,LAct0)=la cat(add(a,la),la1)=add(a,cat(la,la1))
  lower(LAct0,LNat0,n)=LAct0
  lower(add(a,la),add(m,lnat),n)=if(gt(n,m),add(a,lower(la,lnat,n)),lower(la,lnat,n))
  upper(LAct0,LNat0,n)=LAct0
  upper(add(a,la),add(m,lnat),n)=if(gt(n,m),upper(la,lnat,n),add(a,upper(la,lnat,n)))
  mklact(0,a)=LAct0
  mklact(x2p1(n),a)=add(a,cat(mklact(n,a),mklact(n,a)))
  mklact(x2p2(n),a)=add(a,mklact(x2p1(n),a))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Sort ActDT (action or delta or tau)                          %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort ActDT
func
  adt_a:Act->ActDT
  adt_d:->ActDT
  adt_t:->ActDT
map
  eq:ActDT#ActDT->Bool
  if:Bool#ActDT#ActDT->ActDT
  gamma: ActDT#ActDT->ActDT
  annote: Annote#ActDT->ActDT
var
  a,a1:Act adt,adt1:ActDT b:Bool ann:Annote
rew
  eq(adt,adt)=T eq(adt_a(a),adt_a(a1))=eq(a,a1)
  eq(adt_a(a),adt_d)=F eq(adt_a(a),adt_t)=F
  eq(adt_d,adt_a(a))=F eq(adt_d,adt_t)=F
  eq(adt_t,adt_a(a))=F eq(adt_t,adt_d)=F

  if(T,adt,adt1)=adt if(F,adt,adt1)=adt1 if(b,adt,adt)=adt if(not(b),adt,adt1)=if(b,adt1,adt)

  gamma(adt_a(a),adt_a(a1))=if(cannot_communicate(a,a1),adt_d,adt_a(gamma(a,a1)))
  gamma(adt_d,adt)=adt_d gamma(adt_t,adt)=adt_d gamma(adt,adt_d)=adt_d gamma(adt,adt_t)=adt_d

  annote(ann,adt_a(a))=if(in(a,getH(ann)),
                          adt_d,
                          if(in(a,getI(ann)),adt_t,adt_a(appl(a,getR(ann)))))
  annote(ann,adt_d)=adt_d annote(ann,adt_t)=adt_t

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Additional parts for the sorts ALM and AML                   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  getfnd:ALM#LNat->LState             % get the components with indices from LNat
  replfn:ALM#LNat#LALM->ALM           % replace the components with indices from LNat
                                      % with the elements of LALM
  replfn:AML#LNat#LALM->AML
  remfn:ALM#LNat->ALM                 % remove the components with indices from LNat.
  getActDT:ALM#LNat#LAct->ActDT       % get the action a list of ready components
                                      % performing the list of action
  getActDT:AML#LNat#LAct->ActDT       % will communicate into
  is_act:Act#ALM#LNat#LAct->Bool      % is this action a?
  is_tau:ALM#LNat#LAct->Bool          % is this tau?
var
  lm,lm1:ALM ml:AML lnat,lnat1:LNat llm:LALM ann:Annote a,a1:Act la,la1:LAct n,n1:Nat
rew
  getfnd(lm,LNat0)=LState0
  getfnd(lm,add(n,lnat))=add(getf1d(lm,n),getfnd(lm,lnat))

  replfn(lm,add(n,LNat0),add(lm1,LALM0))=replf1(lm,n,lm1)
  replfn(seqM(ml,lm),add(n,lnat),add(lm1,llm))=conp(replfn(ml,add(n,lnat),add(lm1,llm)),lm)

  replfn(AML(lm),lnat,llm)=mkml(replfn(lm,lnat,llm))
  replfn(par(ann,lm,ml),lnat,llm)=
```

```
    annote(ann,comp(if(eq(lower(lnat,lenf(lm)),LNat0),
                     AML(ALM0),
                     mkml(replfn(lm,lower(lnat,lenf(lm)),lower(llm,lnat,lenf(lm))))),
                   if(eq(upper(lnat,lenf(lm)),LNat0),
                     AML(ALM0),
                     replfn(ml,sub(upper(lnat,lenf(lm)),lenf(lm)),upper(llm,lnat,lenf(lm))))))

  remfn(lm,lnat)=replfn(lm,lnat,LEmptyALM(len(lnat)))

  getActDT(lm,add(n,LNat0),add(a,LAct0))=annote(getf1a(lm,n),adt_a(a))
  getActDT(seqM(ml,lm),add(n,add(n1,lnat1)),add(a,add(a1,la1)))=
        getActDT(ml,add(n,add(n1,lnat1)),add(a,add(a1,la1)))

  getActDT(AML(lm),lnat,la)=getActDT(lm,lnat,la)
  getActDT(par(ann,lm,ml),lnat,la)=
        annote(ann,
          if(eq(lower(lnat,lenf(lm)),LNat0),
            getActDT(ml,sub(lnat,lenf(lm)),la),
            if(eq(upper(lnat,lenf(lm)),LNat0),
              getActDT(lm,lnat,la),
              gamma(getActDT(lm,lower(lnat,lenf(lm)),lower(la,lnat,lenf(lm))),
                    getActDT(ml,sub(upper(lnat,lenf(lm)),lenf(lm)),upper(la,lnat,lenf(lm)))))))

  is_act(a,lm,lnat,la)=eq(adt_a(a),getActDT(lm,lnat,la))
  is_tau(lm,lnat,la)=eq(adt_t,getActDT(lm,lnat,la))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%  To be generated from the spec                               %%%
%%%%  The parts that do not parse before actual generation        %%%
%%%%  are commented out                                           %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Communication functions                                     %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  cannot_communicate:Act#Act->Bool
  gamma:Act#Act->Act

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  Functions mkllm_i and f0                                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
map
  mkllm_0:LState#LE_0->LALM
  % f0:ALM#LNat#...#LNat#E_0#E_1#...#E_n->ActPars
var
  d:State ld:LState ee:E_0 lee:LE_0
rew
  mkllm_0(LState0,LE0_0)=add(ALM0,LALM0)
  % mkllm_0(add(d,ld),add(ee,lee))=add([meta(mklm_0)](pr_k(d),pr_k(ee)),mkllm_0(ld,lee))

  % f0(lm,ln_0,...,ln_n,e_0,...,e_n)=
  %     if(not(eq(ln_0,LNat0)),[meta]f_0(getf1(lm,head(ln_0)),e_0),
  %       if(not(eq(ln_1,LNat0)),[meta]f_1(getf1(lm,head(ln_1)),e_1),
  %         if(not(eq(ln_2,LNat0)),[meta]f_2(getf1(lm,head(ln_2)),e_2),
  %           .......
  %             if(not(eq(ln_{n-1},LNat0)),[meta]f_{n-1}(getf1(lm,head(ln_{n-1})),e_{n-1}),
  %               [meta]f_n(getf1(lm,head(ln_n)),e_n)
  %             )
  %           .......
  %         )
  %       )
  %     )
```

C.7 Timed μCRL Code of Auxiliary Data Types

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Sort Time                                                 %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort Time
func
  0:->Time
map
  eq:Time#Time->Bool
  gt:Time#Time->Bool
  min:Time#Time->Time
  max:Time#Time->Time
  if:Bool#Time#Time->Time
rew
  max(t,u)=if(gt(u,t),u,t) min(t,u)=if(gt(t,u),u,t)
  if(T,t,u)=t if(F,t,u)=u if(b,t,t)=t if(not(b),t,u)=if(b,u,t)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Sort E                                                    %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort E
func
  % e:D_0#...#D_n->E
map
  eq: E#E->Bool
  gt: E#E->Bool
  if: Bool#E#E->E
  pr_0:E->Time% pr_1:E->D_1 ... pr_n:E->D_n
  u:E->Time
  c:State#E->Bool
var
  e,e1:E b:Bool
rew
  if(T,e,e1)=e if(F,e,e1)=e1 if(b,e,e)=e if(not(b),e,e1)=if(b,e1,e)
  % gt(e(d0,...,dn),e(e0,...,en))=
  %     or(gt(d0,e0),and(eq(d0,e0),...
  %             or(gt(d{n-1},e{n-1}),and(eq(d{n-1},e{n-1}),gt(dn,en)))...))
  eq(e,e)=T
  % eq(e(d0,...,dn),e(e0,...,en))=and(eq(d0,e0),...,and(eq(dn,en))...)

  % u(e)=pr_0(e)
  % c(d,e)=or(and(not(gt(u(d),[meta]t_0(d,e))),[meta]c_0(d,e)),...
  %          or(and(not(gt(u(d),[meta]t_m(d,e))),[meta]c_m(d,e)),
  %             and(not(gt(u(d),[meta]t_delta(d,e))),[meta]c_delta(d,e)))...)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Sort LE (Lists of E)                                      %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort LE
func
  LE0->LE
  add:E#LE->LE
map
  eq:LE#LE->Bool
  if:Bool#LE#LE->LE
  len:LE->Nat
  head:LE->E

  sat_mmd:Time#LState#LE->Bool
var
  d:State ld:LState e:E le:LE t:Time
rew
  eq(le,le)=T eq(LE0,add(e,le))=F
  eq(add(e,le),LE0)=F eq(add(e,le),add(e1,le1))=and(eq(e,e1),eq(le,le1))
  if(T,le,le1)=le if(F,le,le1)=le1 if(b,le,le)=le if(not(b),le,le1)=if(b,le1,le)
  len(LE0)=0  len(add(e,le))=succ(len(le))  head(add(e,le))=e
  sat_mmd(t,LState0,LE0)=T
  sat_mmd(t,add(d,ld),add(e,le))=
    and(not(gt(t,u(e))),and(c(d,e),sat_mmd(t,ld,le)))
```

Bibliography

[1] Areski Nait Abdallah. *The logic of partial information.* Monographs in TCS. An EATCS Series. Springer, 1995.

[2] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In Bergstra et al. [19], chapter 3, pages 197–292.

[3] R. Alur and D.L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

[4] Suzana Andova. *Probabilistic Process Algebra.* PhD thesis, Eindhoven University of Technology, 2002.

[5] Th. Arts and I.A. van Langevelde. Correct performance of transaction capabilities. In *Proc. 2nd Conference on Applications of Concurrency to System Design (ICACSD'2001)*, pages 35–42, Newcastle upon Tyne, UK, 2001. IEEE Computer Society Press.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, August 1999.

[7] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *TCS*, 51(1/2):129–176, 1987.

[8] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *JACM*, 40(3):653–682, 1993.

[9] J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing.* Monographs in TCS. An EATCS Series. Springer, 2002.

[10] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in TCS 18. Cambridge University Press, 1990.

[11] A.G. Bahmurov, V.I. Chervin, M.V. Chistolinov, J.F. Groote, V.A. Kostenko, R.L. Smeliansky, D.V. Tsarkov, Y.S. Usenko, K. Winter, and V.A. Zakharov. Towards a unified toolset for embedded systems development. *Problems of Programming*, 1(1–2):316–322, 2000. Published in Kyiv, Ukraine.

[12] D.B. Benson and I. Guessarian. Algebraic solutions to recursion schemes. *JCSS*, 35:365–400, 1987.

[13] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[14] J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press, ACM Press Frontier Series, 1989.

[15] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[16] J.A. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In F.R. Drake and J.K. Truss, editors, *Proceedings Logic Colloquium 1986*, pages 21–81, Hull, 1988. North-Holland. First appeared as: Report CS-R8420, CWI, Amsterdam, 1984.

[17] J.A. Bergstra, C.A. Middelburg, and Y.S. Usenko. Discrete time process algebra and the semantics of SDL. In Bergstra et al. [19], chapter 18, pages 1209–1268.

[18] J.A. Bergstra and A. Ponse. Translation of a μCRL-fragment to I-CRL. In *Methods for the Transformation and analysis of CRL. Deliverable 46/SPE/WP5/DS/A/007/b1*, SPECS RACE Project no. 1046, pages 125–148. Available through GSI Tecsi, May 1991.

[19] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[20] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR'94*, LNCS 836, pages 401–416. Springer, 1994.

[21] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: a toolset for analyzing algebraic specifications. In *Proc. CAV'2001*, LNCS 2102, pages 250–254. Springer, July 2001.

[22] S.C.C. Blom and J.C. van de Pol. State space reduction by proving confluence. In E. Brinksma and K.G. Larsen, editors, *Proc. CAV'2002*, LNCS 2404, pages 596–609, July 2002.

[23] D.J.B. Bosscher and A. Ponse. Translating a process algebra with symbolic data values to linear format. In U.H. Engberg, K.G. Larsen, and A. Skou, editors, *Proc. TACAS'95*, Aarhus, Denmark, volume NS-95-2 of *BRICS Notes Series*, pages 119–130. Department of Computer Science, University of Aarhus, May 1995.

[24] J.J. Brunekreef. Process specification in a UNITY format. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing, pages 319–337. Springer, 1995.

[25] Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Verification on infinite structures. In Bergstra et al. [19], chapter 9, pages 545–623.

[26] Stanley N. Burris and H.P. Sankappanavar. *A Course in Universal Algebra.* Number 78 in Graduate Texts in Mathematics. Springer-Verlag, 1981.

[27] K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[28] Søren Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, University of Edinburgh, September 1993.

[29] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT Press, 1999.

[30] R. Cleaveland, G. Lüttgen, and V. Natarajan. Priority in process algebra. In Bergstra et al. [19], chapter 12, pages 711–765.

[31] B. Courcelle. Equivalences and transformations of regular systems–applications to recursive program schemes and grammars. *TCS*, 42:1–122, 1986.

[32] B. Courcelle. Recursive applicative program schemes. In J. van Leeuwen, editor, *Handbook of TCS*, volume B, chapter 9, pages 459–492. Elsevier, 1990.

[33] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles*, pages 238–252, New York, NY, 1977. ACM.

[34] P.J.L. Cuijpers, M.A. Reniers, and W.P.M.H. Heemels. Hybrid transition systems. Technical report, Department of Computer Science. Eindhoven University of Technology, To appear.

[35] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, March 1997.

[36] Dennis Dams and Jan Friso Groote. Specification and implementation of components of a μCRL toolbox. Technical Report 152, Department of Philosophy, Utrecht University, 1995.

[37] P.F.G. Dechering and I.A. van Langevelde. The verification of coordination. In A. Porto and G.-C. Roman, editors, *Proc. 4th Conference on Coordination Languages and Models (COORDINATION'2000)*, LNCS 1906, pages 335–340, Limmasol, Cyprus, 2000. Springer-Verlag.

[38] J.-C. Fernandez, H. Garavel, R. Mateescu A. Kerbrat, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In *Proceedings of the 8th Conference on Computer-Aided Verification*, pages 437–440, New Brunswick, New Jersey, USA, August 1996.

[39] W.J. Fokkink. *Introduction to Process Algebra.* Texts in TCS. An EATCS Series. Springer, 2000.

[40] W.J. Fokkink, N.Y. Ioustinova, E. Kesseler, J.C. van de Pol, Y.S. Usenko, and Y.A. Yushtein. Refinement and verification applied to an in-flight data acquisition unit. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *Proc. CONCUR'2002*, LNCS 2421, pages 1–23. Springer, August 2002.

[41] Hubert Garavel. *Compilation et Vérification de Programmes LOTOS*. PhD thesis, Université Joseph Fourier (Grenoble), November 1989. In French.

[42] Hubert Garavel. CAESAR reference manual. Rapport SPECTRE C18, Laboratoire de Génie Informatique - Institut IMAG, Grenoble, November 1990.

[43] Hubert Garavel and Joseph Sifakis. Compilation and verification of LOTOS specifications. In L. Logrippo, R.L. Probert, and H. Ural, editors, *Proc. 10th International Symposium on Protocol Specification, Testing and Verification*, pages 379–394, Ottawa, Canada, June 1990.

[44] R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. Manuscript. Preliminary version available by ftp at `ftp://boole.stanford.edu/pub/spectrum.ps.gz`, 1993. Extended abstract in E. Best, editor: Proceedings *CONCUR'93*, Hildesheim, Germany, August 1993, LNCS 715, Springer, pp. 66–81.

[45] R.J. van Glabbeek. The linear time - branching time spectrum I. In Bergstra et al. [19], chapter 1, pages 3–99.

[46] S.A. Greibach. A new normal-form theorem for context-free phase structure grammars. *JACM*, 12(1):42–52, 1965.

[47] J.F. Groote. The syntax and semantics of timed μCRL. Report SEN-R9709, CWI, Amsterdam, 1997.

[48] J.F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. In M. Leuschel and U. Ultes-Nitsche, editors, *Proc. 3rd International Workshop on Verification and Computational Logic (VCL2002)*, Technical Report DSSE-TR-2002-5. Department of Electronics and Computer Science, University of Southampton, October 2002.

[49] J.F. Groote and S.P. Luttik. Undecidability and completeness results for process algebras with alternative quantification over data. Report SEN-R9806, CWI, Amsterdam, July 1998. Available from `http://www.cwi.nl/~luttik/`.

[50] J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In Armando Martin Haeberer, editor, *Proc. of AMAST'98 (Amazonia, Brazil)*, LNCS 1548, pages 74–90. Springer-Verlag, 1998.

[51] J.F. Groote, J. Pang, and A.G. Wouters. Analysis of a distributed system for lifting trucks. *JLAP*, 2003. To appear.

[52] J.F. Groote and A. Ponse. Proof theory for μCRL: A language for processes with data. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Semantics of Specification Languages*, Workshop in Computing, pages 232–251. Springer, 1994.

[53] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes 1994*, Workshop in Computing, pages 26–62. Springer, 1995.

[54] J.F. Groote, A. Ponse, and Y.S. Usenko. Linearization in parallel pCRL. *JLAP*, 48(1-2):39–72, June 2001.

[55] J.F. Groote, M.A. Reniers, J.J. van Wamel, and M.B. van der Zwaag. Completeness of timed μCRL. *Fundamenta Informaticæ*, 50(3-4):361–402, 2002.

[56] J.F. Groote and M.P.A. Sellink. Confluence for process verification. *TCS*, 170(1–2):47–81, 1996.

[57] J.F. Groote and J. Springintveld. Focus points and convergent process operators. A proof strategy for protocol verification. *JLAP*, 49:31–60, 2001.

[58] J.F. Groote and J.J. van Wamel. Analysis of three hybrid systems in timed μCRL. *SCP*, 39:215–247, 2001.

[59] J.F. Groote and J.J. van Wamel. Algebraic data types and induction in μCRL. Report P9409, University of Amsterdam, Programming Research Group, 1994.

[60] J.F. Groote and J.J. van Wamel. The parallel composition of uniform processes with data. *TCS*, 266(1-2):631–652, 2001.

[61] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality.* LNCS 2428. Springer, 2002.

[62] Y. Hirshfeld and F. Moller. Decidability results in automata and process theory. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 102–148, Berlin, 1996. Springer.

[63] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[64] J. Hooman and J.C. van de Pol. Formal verification of replication on a distributed data space architecture. In *Proc. 17th Symposium on Applied Computing (SAC'2002) — Coordination Models, Languages and Applications*, pages 351–358, Madrid, Spain, 2002. ACM Press.

[65] Anna Ingólfsdóttir and Huimin Lin. A symbolic approach to value-passing processes. In Bergstra et al. [19], chapter 7, pages 427–478.

[66] ISO/IEC. *LOTOS — a formal description technique based on the temporal ordering of observational behaviour.* International Standard 8807. International Organization for Standardization, — Information Processing Systems — Open Systems Interconection, Genève, September 1988.

[67] B. Jonsson, Wang Yi, and K.G. Larsen. Probabilistic extensions of process algebras. In Bergstra et al. [19], chapter 11, pages 685–710.

[68] J. Loeckx, H.-D. Ehrich, and M. Wolf. Algebraic specification of abstract data types. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol 5*, chapter 4, pages 217–316. Oxford University Press, 2000.

[69] Bas Luttik. *Choice Quantification in Process Algebra.* PhD thesis, Universiteit van Amsterdam, 2002.

[70] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[71] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. *SCP*, 2002.

[72] S. Mauw and J.C. Mulder. Regularity of BPA-systems is decidable. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR '94*, LNCS 836, pages 34–47. Springer, 1994.

[73] S. Mauw and G.J. Veltink. A process specification formalism. *Fund. Inf.*, XIII:85–139, 1990.

[74] S. Mauw and G.J. Veltink, editors. *Algebraic Specification of Communication Protocols.* Cambridge Tracts in TCS 36. Cambridge University Press, 1993.

[75] K. Meinke and J.V. Tucker. Universal algebra. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol 1*, chapter 3, pages 189–411. Oxford University Press, 1993.

[76] C.A. Middelburg. Variable binding operators in transition system specifications. *JLAP*, 47(1):15–45, 2001.

[77] R. Milner. A complete inference system for a class of regular behaviours. *JCSS*, 28:439–466, 1984.

[78] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[79] V. van Oostrom. Personal communications, 2000.

[80] J. Pang. Analysis of a security protocol in μCRL. In *Proc. 4th Conference on Formal Engineering Methods*, LNCS 2495, pages 396–400. Springer, 2002.

[81] Joachim Parrow. An introduction to the π-calculus. In Bergstra et al. [19], chapter 8, pages 479–543.

[82] D.A. Peled, V.R. Pratt, and G.J. Holzmann, editors. *Partial Order Methods in Verification*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, July 1997.

[83] J.C. van de Pol. A prover for the μCRL toolset with applications – version 0.1. Report SEN-R0106, CWI, Amsterdam, April 2001.

[84] J.C. van de Pol. JITty: a rewriter with strategy annotations. In *Proc. RTA 2002*, Copenhagen, 2002.

[85] J.C. van de Pol and M. Valero Espada. Formal specification of JavaSpaces (TM) architecture using μCRL. In F. Arbab and C.L. Talcott, editors, *Proc. 5th Conference on Coordination Languages and Models (COORDINATION'2002)*, LNCS 2315, pages 274–290, York, UK, 2002. Springer-Verlag.

[86] A. Ponse and Y.S. Usenko. Equivalence of recursive specifications in process algebra. *IPL*, 80(1):59–65, October 2001.

[87] Alban Ponse. Computable processes and bisimulation equivalence. *Formal Aspects of Computing*, 8(6):648–678, 1996.

[88] Helena Rasiowa and Roman Sikorski. *The Mathematics of Metamathematics.* P.W.N., Warszawa, third edition, 1970.

[89] M.A. Reniers and Y.S. Usenko. Linearization of timed μCRL specifications. Unpublished, 2002.

[90] Albert Rubio. A fully syntactic AC-RPO. *Information and Computation*, 2002. To appear.

[91] Konstantin O. Savenkov. Linearization of algebraic specifications using rewrite rules. Master's thesis, Moscow State University, 2002. In Russian.

[92] SPECS-Semantics and Analysis. *Definition of MR and CRL Version 2.1. Deliverable 46/SPE/WP5/DS/A/017/b1.* SPECS RACE Project no. 1046. Available through GSI Tecsi, 1990.

[93] Colin Stirling. *Modal and Temporal Properties of Processes.* Texts in CS. Springer, 2001.

[94] Yong Sun. An algebraic generalization of Frege structures — binding algebras. *TCS*, 211:189–232, 1999.

[95] Y.S. Usenko. Linearization of μCRL specifications. Report SEN-R0209, CWI, Amsterdam, 2002.

[96] Y.S. Usenko. Linearization of μCRL specifications (extended abstract). In M. Leuschel and U. Ultes-Nitsche, editors, *Proc. 3rd International Workshop on Verification and Computational Logic (VCL2002)*, Technical Report DSSE-TR-2002-5. Department of Electronics and Computer Science, University of Southampton, October 2002.

[97] Y.S. Usenko. State space generation for the HAVi leader election protocol. *SCP*, 43:1–33, April 2002.

[98] Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In A. Middeldorp, editor, *Proc. RTA'01*, LNCS 2051, pages 357–361. Springer, May 2001.

[99] J.J. van Wamel. *Verification Techniques for Elementary Data Types and Retransmission Protocols.* PhD thesis, Universiteit van Amsterdam, September 1995.

[100] Freek Wiedijk. *Persistence in Algebraic Specifications.* PhD thesis, Universiteit van Amsterdam, December 1991.

[101] T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* PhD thesis, Eindhoven University of Technology, 2003.

[102] A.G. Wouters. Manual for the μCRL tool set (version 2.8.2). Report SEN-R0130, CWI, Amsterdam, December 2001.

[103] M.B. van der Zwaag. The cones and foci proof technique for timed transition systems. *IPL*, 80(1):33–40, 2001.

[104] Specification and description language (SDL). ITU-T Recommendation Z.100, 1994.

Summary

In this thesis a linearization algorithm for transforming an arbitrary guarded μCRL specification to a linear form is presented. This linear form (called *Linear Process Equation (LPE)*) is a simple and constructive symbolic representation of a labeled transition system. The LPE format is the core format used in the tools for μCRL.

First, the simple case of parallel pCRL processes is considered. It does not include processes with recursive parallelism, e.g. systems where processes can be created dynamically. Next, the whole μCRL case is considered, and a specifically defined algebraic data type is used to model parallel and sequential compositions of processes. Finally, the case of timed μCRL is considered. As a result of timed μCRL linearization, the existing tools for untimed μCRL can, in principle, be used to analyze timed systems.

In order to prove correctness of the linearization algorithms, a new notion of equivalence is defined for systems of equations and recursive program schemata in process algebra. Two systems are equivalent if in any model of μCRL they have the same set of solutions. The syntactic counterpart of this notion is defined using a similar kind of equational calculus as used for μCRL. The new equivalence does not depend on the number of solutions a system has in a particular model of μCRL.

Samenvatting

In dit proefschrift presenteren we een linearisatie algoritme voor het transformeren van een willekeurige guarded μCRL specificatie naar een corresponderende lineaire vorm. Zo'n lineaire vorm wordt een *lineaire procesvergelijking* genoemd. Lineaire procesvergelijkingen zijn simpele, constructieve symbolische representaties van gelabelde transitiesystemen. Vanwege de restricties op hun vorm zijn lineaire procesvergelijkingen geschikt voor automatische manipulatie. Ze vormen dan ook de kern van de μCRL tools.

We beschouwen drie gevallen van linearisatie. Allereerst beschouwen we linearisatie voor het eenvoudige geval van parallelle pCRL processen. Deze processen hebben geen recursief parallellisme, d.w.z. processen kunnen niet dynamisch gecreëerd worden. Vervolgens breiden we linearisatie uit tot alle μCRL processen. Voor dat doel wordt een algebraïsch datatype geïntroduceerd dat gebruikt wordt om de parallelle en sequentiële compositie van processen te modelleren. Ten slotte presenteren we een linearisatie algoritme voor Timed μCRL. Als een direct gevolg van dit laatste algoritme kunnen bestaande μCRL tools in principe gebruikt worden voor de analyse van systemen met tijd.

Om de correctheid van de linearisatie algoritmes te bewijzen, definiëren we een nieuwe notie van equivalentie voor systemen van vergelijkingen en recursieve programma schema's in procesalgebra. We zeggen dat twee systemen equivalent zijn als ze dezelfde verzameling van oplossingen hebben in ieder model van μCRL. Om een met deze semantische notie corresponderende syntactische notie van equivalentie te definiëren, maken we gebruik van een equationele calculus, analoog aan de bestaande equationele calculus voor μCRL. Een onderscheidende eigenschap van onze notie van equivalentie is dat zij niet afhangt van het aantal oplossingen dat een systeem heeft in een speciaal model van μCRL.

Curriculum Vitæ

22 October 1975 Born in Kyiv, Ukraine.

September 1990 – June 1992 Natural Science Kyiv Lyceum #145.

September 1992 – June 1997 Taras Shevchenko University of Kyiv, Faculty of Cybernetics. Master's degree in applied mathematics, specialization in theoretical cybernetics.

September 1996 – May 1997 Fellow at the United Nations University / International Institute for Software Technology (UNU/IIST) in Macau.

October 1997 – July 1998 PhD student at Research Institute for Symbolic Computations (RISC), Johannes-Kepler University, Linz, Austria.

September 1998 – August 2002 Project researcher at Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.

From September 2002 PostDoc researcher at Formal Methods and Tools (FMT) Group, University of Twente, Enschede, The Netherlands.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1996-1

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms.* Faculty of Mathematics and Computer Science, KUN. 1996-2

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation.* Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. *Parallel Local Search.* Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory.* Faculty of Mathematics and Computer Science, KUN. 1996-5

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems.* Faculty of Mathematics and Computing Science, TUE. 1996-6

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance.* Faculty of Mathematics and Computer Science, UvA. 1996-7

H. Doornbos. *Reductivity Arguments and Program Construction.* Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. *Functorial Operational Semantics and its Denotational Dual.* Faculty of Mathematics and Computer Science, VUA. 1996-9

A.M.G. Peeters. *Single-Rail Handshake Circuits.* Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism.* Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference.* Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics.* Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth.* Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context.* Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types.* Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics.* Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution.* Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing.* Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing.* Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering.* Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation.* Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller.* Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model.* Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing.* Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes.* Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A*

Case-study into Acute Effects of Air Pollution Episodes. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ.* Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16