Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Relating State Transformation Semantics and Predicate Transformer Semantics for Parallel Programs

Franck van Breugel

Computer Science/Department of Software Technology

CS-R9339 1993

Relating State Transformation Semantics and Predicate Transformer Semantics for Parallel Programs

Franck van Breugel

CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Vrije Universiteit
P.O. Box 7161, 1007 MC Amsterdam, The Netherlands

Abstract

A state transformation semantics and a predicate transformer semantics for programs built from atomic actions, sequential composition, nondeterministic choice, parallel composition, atomisation, and recursion are presented. Both semantic models are derived from some SOS-style labelled transition system. The state transformation semantics and the predicate transformer semantics are shown to be isomorphic extending results of Plotkin and Best.

AMS Subject Classification (1991): 68Q55
CR Subject Classification (1991): D.3.1, F.3.2
Keywords & Phrases: state transformation, predicate transformer, isomorphism, labelled transition system, parallelism
Note: This work was partially supported by the Netherlands *Nationale Faciliteit Informatica* programme, project Research and Education in Concurrent Systems (REX).

Introduction

In [Dij76], Dijkstra introduces a *predicate transformer semantics*, which is called the weakest precondition semantics, to deal with partial correctness of sequential programs. The relation between this predicate transformer semantics and a *state transformation semantics* is considered by De Roever in [Roe76]. In [Bak77], De Bakker shows that there is a *homomorphism* from the state transformation semantics to the predicate transformer semantics. Plotkin shows, in [Plo79], that by refining the definitions this homomorphism can be strengthened to an *isomorphism*.

Predicate transformer semantics for partial correctness of *parallel programs* are studied by Van Lamsweerde and Sintzoff [LS79], Haase [Haa81], Flon and Suzuki [FS81], Elrad and Francez [EF84], Best [Bes82, Bes89], and Scholefield and Zedan [SZ92]. In [Bes89], a predicate transformer semantics and a state transformation semantics are related. However, only parallel programs without recursion are considered.

In this paper, we present a state transformation semantics and a predicate transformer semantics for parallel programs built from atomic actions, sequential composition, nondeterministic choice, parallel composition, atomisation (atomicity being a key notion in reasoning about parallel programs, see, e.g., [LL90]), and recursion. (In order to introduce recursion we will consider a program to be a statement and a declaration. The statement is built from atomic actions, procedure variables, and the operators mentioned above, and the declaration assigns to the procedure variables their corresponding bodies, which are again statements. In general, we will fix the declaration part of a program and only consider the statement part.) These semantics are shown to be isomorphic. Although several results of [Plo79] are exploited in our paper, for parallel programs the isomorphism result cannot be obtained simply along the lines of [Plo79], because the parallel composition is not compositional with respect to the (state transformation and predicate transformer) semantics defined in [Plo79]. The standard example to show that compositionality is lost when introducing parallel composition is that the programs

$$x := 2 \text{ and } x := 1\,; x := x + 1$$

are semantically equivalent, but

$$x := 2 \parallel x := 3 \text{ and } (x := 1\,; x := x + 1) \parallel x := 3$$

are not (see, e.g., [Mil93]).

The state transformation semantics and the predicate transformer semantics to be presented are mappings from statements to state transformations and predicate transformers, respectively. Before we come to these semantics, we first discus the state transformations and the predicate transformers and their relationship.

To model a statement we can use a *state transformation* assigning to an arbitrary (initial) state the result of the statement started in the state. If the statement (always) terminates, then the result is described by a set of (possible final) states. (Note that we need *sets* of states to model nondeterminism.) Otherwise, i.e. if the statement might not terminate, the result is modelled by a special element. For the time being, the set of state transformations can be viewed as

$$(\alpha \in)\, \Sigma \rightarrow (\mathcal{P}(\Sigma) \cup \{\Sigma_\perp\}),$$

with $\Sigma_\perp$ being the above mentioned special element.

A statement can also be modelled by a *predicate transformer*. A predicate transformer transforms a predicate valid after the execution of a statement to a predicate valid before the execution of the statement. (Note the order reversal when we go from state transformations to predicate transformers in that a state transformation maps an initial state to a set of final states whereas a predicate transformer maps a 'final' predicate to an 'initial' predicate.) To model predicates we employ sets of states. (Taking $\mathcal{P}(\Sigma)$ as the set of predicates is the so-called extensional view. We could also have taken the so-called intensional view by describing the set of predicates by $\Sigma \rightarrow \{true, false\}$ as is done in, e.g., [Bak80].) Predicate transformers will look like

$$(\beta \in)\, \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma).$$

With each state transformation α we can associate a predicate transformer $\omega(\alpha)$ defined by

$$\omega(\alpha) = \lambda S \cdot \{\, \sigma \in \Sigma \mid \alpha(\sigma) \subseteq S \,\}.$$

The set $\omega(\alpha)(S)$ is called the *weakest precondition* of S relative to α (cf. [Roe76]). By refining the above definitions of state transformations and predicate transformers and by endowing them with orders, complete partial orders of state transformations and predicate transformers can be obtained. These complete partial orders can be shown to be isomorphic (by means of a modification of ω and its inverse). These refinements and endowments with orders can be done in various ways. Here, we will follow [Plo79]. Other isomorphism results of state transformations and predicate transformers are presented by Wand [Wan77], Majster-Cederbaum [MC80], Best [Bes82], Smyth [Smy83, Smy92], Apt and Plotkin [AP86], and Bonsangue and Kok [BK92, BK93]. For an overview of these isomorphism results we refer the reader to [BK92].

Having discussed the state transformations and the predicate transformers, we now come to the state transformation and predicate transformer semantics. Both semantic models are so-called *operational semantics* defined by means of a *labelled transition system* (cf. [Plo81, Plo82]). Let us first discus the state transformation semantics. The labels of the labelled transition system defining the state transformation semantics are state transformations. To deal with recursion, we employ pairs of statements and natural numbers as configurations of the labelled transition system. A configuration (s, n) denotes that statement s has to be executed with recursion restricted to at most depth n. By means of the labelled transition system we define an (intermediate) state transformation semantics $\mathcal{O}^*_{st}$ mapping a configuration to a state transformation. This state transformation is obtained by composing the labels of the transition sequences starting from the configuration. The state transformation semantics $\mathcal{O}_{st}$ maps statements to state transformations. For statement s, $\mathcal{O}_{st}(s)$ is defined as the least upper bound (in the complete partial order of state transformations) of the $\mathcal{O}^*_{st}(s, n)$'s. The predicate transformer semantics $\mathcal{O}_{pt}$ is defined similar to the state transformation semantics $\mathcal{O}_{st}$. In this case, the labels of the labelled transition system are predicate transformers. These two step definitions are choosen to smoothen the proof of the theorem establishing the isomorphism of the state transformation semantics and the predicate transformer semantics.

The main result of this paper is the relation of the state transformation semantics $\mathcal{O}_{st}$ and the predicate transformer semantics $\mathcal{O}_{pt}$. We will show that these semantics are isomorphic. A problem in the proof of this theorem is that the state transformation semantics $\mathcal{O}_{st}$ and the predicate transformer semantics $\mathcal{O}_{pt}$ are not compositional. As an intermediate tool, *compositional semantics* $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are introduced. The semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are defined by means of the labelled transition systems defining the semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{pt}$, respectively. This time, the semantics assign to a configuration a set of *sequences* of state transformations and predicate transformers (cf. [Coo78]). To a configuration the set of label sequences corresponding to the set of transition sequences starting from the configuration is assigned. Using sets of sequences of state transformations and predicate transformers instead of state transformations and predicate transformers gives rise to compositional semantics as we will see. We prove that the compositional semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are isomorphic. For this purpose, we use the already mentioned result of [Plo79] that a complete partial order of state transformations (ST) and a complete partial order of predicate transformers (PT) are isomorphic, $\omega : ST \cong PT$. From this result, we can easily derive that the sets of (nonempty and finite) sets of (nonempty and finite) sequences of state transformations and predicate transformers are isomorphic, $\Omega : \mathcal{P}_{nf}(ST^+) \cong \mathcal{P}_{nf}(PT^+)$. Based on this, the compositional semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are shown to be isomorphic. This proof shows some resemblance with the isomorphism proof in [Bes89].

$$
\begin{tikzcd}
 & Conf \arrow[dl, "\mathcal{C}_{st}"'] \arrow[dr, "\mathcal{C}_{pt}"] \arrow[d, phantom, "*"] & \\
\mathcal{P}_{nf}(ST^+) \arrow[rr, leftrightarrow, "\Omega"'] & & \mathcal{P}_{nf}(PT^+)
\end{tikzcd}
$$

From this isomorphism result we will derive the isomorphism of $\mathcal{O}_{st}$ and $\mathcal{O}_{pt}$ as follows. We introduce abstraction operators abs_{st} and abs_{pt}. These operators map sets of sequences of state transformations and predicate transformers to state transformations and predicate transformers, respectively. The sets of sequences will be composed as in the definitions of the intermediate semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{pt}$.

First, we relate the isomorphisms ω and Ω by means of the abstraction operators.

$$
\begin{tikzcd}
\mathcal{P}_{nf}(ST^{+}) \arrow[r, leftrightarrow, "\Omega"] \arrow[d, "abs_{st}"'] \arrow[dr, phantom, "*"] & \mathcal{P}_{nf}(PT^{+}) \arrow[d, "abs_{pt}"] \\
ST \arrow[r, leftrightarrow, "\omega"'] & PT
\end{tikzcd}
$$

Second, we relate the compositional semantics with the intermediate semantics.

$$
\begin{tikzcd}
\mathcal{P}_{nf}(ST^{+}) \arrow[dd, "abs_{st}"'] & & & & \mathcal{P}_{nf}(PT^{+}) \arrow[dd, "abs_{pt}"] \\
& * & Conf \arrow[ull, "\mathcal{C}_{st}"'] \arrow[urr, "\mathcal{C}_{pt}"] \arrow[dll, "\mathcal{O}^{*}_{st}"] \arrow[drr, "\mathcal{O}^{*}_{pt}"'] & * & \\
ST & & & & PT
\end{tikzcd}
$$

By combining the above results, we finally show that the state transformation semantics $\mathcal{O}_{st}$ and the predicate transformer semantics $\mathcal{O}_{pt}$ are isomorphic.

$$
\begin{tikzcd}
& Stat \arrow[dl, "\mathcal{O}_{st}"'] \arrow[dr, "\mathcal{O}_{pt}"] \arrow[d, phantom, "*"] & \\
ST \arrow[rr, leftrightarrow, "\omega"'] & & PT
\end{tikzcd}
$$

In the first section of this paper, some results of [Plo79] are repeated. A complete partial order of state transformations and a complete partial order of predicate transformers, which are isomorphic, are introduced. In the second section, the state transformation semantics and the predicate transformer semantics are presented. The paper concludes with the theorem establishing the isomorphism of these semantics. The theorem extends the isomorphism results of [Plo79] and [Bes89] by going from sequential programs to parallel programs and by adding recursion, respectively.

Acknowledgements

The author would like to thank Jaco de Bakker for introducing him to the problem of relating state transformation semantics and predicate transformer semantics for parallel programs and discussing the topic. Furthermore, the author is grateful to Marcello Bonsangue, Wim Hesselink, Joost Kok, Vincent van Oostrom, Prakash Panangaden, Jan Rutten, Daniele Turi, and Erik de Vink for their comments on a preliminary version of this paper. Finally, the author is thankful to one of the CONCUR referees for pointing out that a previous definition of the complexity measure (Definition 2.5) did not validate Property 2.6.

1. State transformations and predicate transformers

In this section, some results of [Plo79] are repeated. A complete partial order of state transformations and a complete partial order of predicate transformers are introduced and shown to be isomorphic.

1.1 State transformations

First, we define a complete partial order of state transformations (cf. Definitions 5.1, 5.2, and 5.3 of [Plo79]). We postulate a denumerable set $(\sigma \in)\, \Sigma$ of *states* (cf. page 212 of [Wan77] and page 529 of [Plo79]). State transformations are defined as mappings from Σ to the so-called Smyth power domain ([Smy78]) of Σ in

Definition 1.1 The partial order $(\alpha \in)\, ST$ of *state transformations* is the set

$$\Sigma \to \mathcal{P}_S\,(\Sigma_\perp),$$

where

$$\mathcal{P}_S\,(\Sigma_\perp) = \{\, S \subseteq \Sigma \mid S \text{ is nonempty and finite} \,\} \cup \{\Sigma_\perp\}$$

and $\Sigma_\perp$ denotes the disjoint union of Σ and $\{\perp\}$. This set is ordered by $\alpha \sqsubseteq \alpha'$ if

for all $\sigma \in \Sigma$, $\alpha\,(\sigma) \supseteq \alpha'\,(\sigma)$.

Property 1.2 *ST is a complete partial order.*

Proof See Proposition 5.4 of [Plo79]. □

Composition and union of state transformations are defined in

Definition 1.3 The mapping $\bullet : ST \times ST \to ST$ is defined by

$$\alpha \bullet \alpha' = \lambda\sigma \cdot \begin{cases} \bigcup \{\, \alpha'\,(\sigma') \mid \sigma' \in \alpha\,(\sigma) \,\} & \text{if } \alpha\,(\sigma) \neq \Sigma_\perp \\ \Sigma_\perp & \text{otherwise} \end{cases}$$

and the mapping $\cup : ST \times ST \to ST$ is defined by

$$\alpha \cup \alpha' = \lambda\sigma \cdot \alpha\,(\sigma) \cup \alpha'\,(\sigma).$$

Note that the state transformation $\alpha \bullet \alpha'$ is the result of performing state transformation α' after state transformation α. The definition of the composition is similar to the definition of *Comp* at page 543 of [Plo79]. The definition of the union is similar to the definition of $\sqcap$ at page 542 of [Plo79]. We will use the composition and union of state transformations to compose the labels, i.e. state transformations, of transition sequences of the labelled transition system in the definition of the (intermediate) state transformation semantics.

Property 1.4 *The mappings $\bullet$ and $\cup$ are strict in both arguments.*

Proof Trivial. □

The strictness of composition and union will be applied in the well-definedness proof of the state transformation semantics.

1.2 Predicate transformers

Second, we define a complete partial order of predicate transformers (cf. page 537 of [Plo79]).

Definition 1.5 The partial order $(\beta \in)\, PT$ of *predicate transformers* is the set

$$\{\, \beta \in \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma) \mid \beta \text{ is strict, continuous, and multiplicative} \,\},$$

where β is called *multiplicative* if

for all $S, S' \in \mathcal{P}(\Sigma)$, $\beta(S \cap S') = \beta(S) \cap \beta(S')$.

This set is ordered by $\beta \sqsubseteq \beta'$ if

for all $S \in \mathcal{P}(\Sigma)$, $\beta(S) \subseteq \beta'(S)$.

Property 1.6 *PT is a complete partial order.*

Proof See Proposition 4.2 of [Plo79]. □

For predicate transformers, composition and intersection are defined in

Definition 1.7 The mapping $\bullet : PT \times PT \rightarrow PT$ is defined by

$$\beta \bullet \beta' = \lambda S \cdot \beta'(\beta(S))$$

and the mapping $\cap : PT \times PT \rightarrow PT$ is defined by

$$\beta \cap \beta' = \lambda S \cdot \beta(S) \cap \beta'(S).$$

The predicate transformer $\beta \bullet \beta'$ is the result of performing predicate transformer β' after predicate transformer β. The definition of the composition is the 'reversed' version of *Comp* at page 538 of [Plo79], i.e. $\beta \bullet \beta' = Comp(\beta, \beta')$. The definition of the intersection is similar to the definition of $\sqcap$ at page 537 of [Plo79]. The composition and intersection of state transformations will be used to compose the labels, i.e. predicate transformers, of transition sequences of the labelled transition system in the definition of the (intermediate) predicate transformer semantics.

Property 1.8 *The mappings $\bullet$ and $\cap$ are strict in both arguments.*

Proof Trivial. □

The strictness of composition and intersection will be exploited in the well-definedness proof of the predicate transformer semantics.

1.3 Isomorphism theorem

Third, we show ST and PT to be isomorphic. To define the isomorphism the following *stability lemma* (this term originates from the term stable function which has been introduced in [Ber78]) is proved.

Lemma 1.9 *Let $\beta \in \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)$. Then $\beta \in PT$ if and only if whenever $\sigma \in \beta(\Sigma)$ there exists a nonempty and finite set $min(\beta, \sigma)$ satisfying* *for all* $S \in \mathcal{P}(\Sigma), \sigma \in \beta(S) \Leftrightarrow \mathit{min}(\beta, \sigma) \subseteq S$.

PROOF See Lemma 5.6 of [Plo79]. □

The isomorphism ω and its inverse ω^{-1} are defined as follows (cf. Definition 5.8 of [Plo79]).

DEFINITION 1.10 The mapping $\omega : ST \to PT$ is defined by

$$\omega(\alpha) = \lambda S \cdot \{\, \sigma \in \Sigma \mid \alpha(\sigma) \subseteq S \,\}$$

and the mapping $\omega^{-1} : PT \to ST$ is defined by

$$\omega^{-1}(\beta) = \lambda\sigma \cdot \begin{cases} \mathit{min}(\beta, \sigma) & \text{if } \sigma \in \beta(\Sigma) \\ \Sigma_{\perp} & \text{otherwise} \end{cases}$$

THEOREM 1.11 *The mapping* $\omega : ST \cong PT$ *is an isomorphism of complete partial orders.*

PROOF See Theorem 5.9 of [Plo79]. □

The predicate transformer corresponding to the composition of the state transformations α and α' is the composition of the predicate transformers corresponding to the state transformations α' and α (order reversal). The predicate transformer corresponding to the union of the state transformations α and α' is the intersection of the predicate transformers corresponding to the state transformations α and α'.

PROPERTY 1.12 *For all* α *and* α',

$$\begin{aligned} \omega(\alpha \bullet \alpha') &= \omega(\alpha') \bullet \omega(\alpha) \\ \omega(\alpha \cup \alpha') &= \omega(\alpha) \cap \omega(\alpha') \end{aligned}$$

PROOF See Lemmas 5.10 and 5.11 of [Plo79]. □

The proof that the state transformation semantics and the predicate transformer semantics are isomorphic, will be based on these properties.

2. State transformation semantics and predicate transformer semantics

In this section, we present the state transformation semantics and the predicate transformer semantics. After we have introduced these semantic models, we head for the main theorem of this paper. In this theorem, we show that the semantics are isomorphic.

The parallel programs are built from atomic actions a, sequential composition ;, nondeterministic choice $+$, parallel composition $\|$, atomisation $[\ ]$, and procedure variables x.

DEFINITION 2.1 The class $(s \in)\ \mathit{Stat}$ of *statements* is defined by

$$s ::= a \mid s \,;\, s \mid s + s \mid s \parallel s \mid [s] \mid x$$

and the class $(d \in)\ \mathit{Decl}$ of *declarations* is defined by

$$\mathit{Decl} = \mathit{PVar} \to \mathit{Stat}$$

and the class $(p \in)$ *Prog* of *programs* is defined by

$$Prog = Decl \times Stat.$$

In the sequel, we fix the declaration part of a program and only consider the statement part.

2.1 State transformation semantics

First, we present the state transformation semantics. The semantics is defined by means of a labelled transition system. In the labelled transition system, we employ configurations (s, n) in $Stat \times I\!N$. The natural number n in the configuration (s, n) denotes the maximal depth of recursion which is allowed during the execution of statement s. The use of such configurations will facilitate the subsequent proof that the state transformation semantics and the predicate transformer semantics are isomorphic. The configurations of the labelled transition system are defined in

Definition 2.2 The class $(c \in)$ *Conf* of *configurations* is defined by

$$c ::= (s, n) \mid c\,; c \mid c + c \mid c \parallel c \mid [c].$$

Furthermore, the empty statement E, which denotes termination, is used as configuration of the labelled transition system. The labels of the labelled transition system are state transformations. We presuppose a mapping $\mathcal{ST}$ assigning to each atomic action a state transformation. For an atomic action a and a state σ, $\mathcal{ST}(a)(\sigma)$ is the singleton set consisting of the state after the execution of a started in σ. The transition relation of the labelled transition system is defined in

Definition 2.3 The transition relation $\rightarrow$ is the smallest subset of

$$(Conf \cup \{\mathrm{E}\}) \times ST \times (Conf \cup \{\mathrm{E}\})$$

satisfying

- $(a, n) \xrightarrow{\mathcal{ST}(a)} \mathrm{E}$
- $(x, 0) \xrightarrow{\lambda\sigma\cdot\Sigma_\perp} \mathrm{E}$
- $\dfrac{(s_1, n)\,;(s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}{(s_1\,; s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}$
- $\dfrac{(s_1, n) + (s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}{(s_1 + s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}$
- $\dfrac{(s_1, n) \parallel (s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}{(s_1 \parallel s_2, n) \xrightarrow{\alpha} \mathrm{E} \mid c}$
- $\dfrac{[(s, n)] \xrightarrow{\alpha} \mathrm{E} \mid c}{([s], n) \xrightarrow{\alpha} \mathrm{E} \mid c}$
- $\dfrac{(d(x), n) \xrightarrow{\alpha} \mathrm{E} \mid c}{(x, n+1) \xrightarrow{\alpha} \mathrm{E} \mid c}$
- $\dfrac{c_1 \xrightarrow{\alpha} \mathrm{E} \mid c_1'}{c_1\,; c_2 \xrightarrow{\alpha} c_2 \mid c_1'\,; c_2}$
- $\dfrac{c_1 \xrightarrow{\alpha} \mathrm{E} \mid c_1'}{\begin{array}{l} c_1 + c_2 \xrightarrow{\alpha} \mathrm{E} \mid c_1' \\ c_2 + c_1 \xrightarrow{\alpha} \mathrm{E} \mid c_1' \end{array}}$
- $\dfrac{c_1 \xrightarrow{\alpha} \mathrm{E} \mid c_1'}{\begin{array}{l} c_1 \parallel c_2 \xrightarrow{\alpha} c_2 \mid c_1' \parallel c_2 \\ c_2 \parallel c_1 \xrightarrow{\alpha} c_2 \mid c_2 \parallel c_1' \end{array}}$

- $$\frac{c \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E}}{[c] \xrightarrow{\alpha_1 \bullet \cdots \bullet \alpha_{k+1}} \mathrm{E}}$$

In the configuration $(x, 0)$ only recursion at depth 0, i.e. no recursion, is allowed. Consequently, the procedure variable x cannot be replaced by its declaration $d(x)$. In this case, $(x, 0)$ can only do a $\lambda\sigma \cdot \Sigma_\perp$-step, which denotes nontermination (see Introduction). In the configuration $(x, n+1)$, the procedure variable x can be replaced by its declaration $d(x)$. However, in $d(x)$ only recursion at depth at most n is allowed.

Although the parallel composition fits in the framework described above, for example, the conditional statement does not fit in this framework, because the obvious transition rule

- $$\frac{c_1 \xrightarrow{\alpha} \mathrm{E} \mid c_1'}{\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \xrightarrow{\alpha'} \mathrm{E} \mid c_1'} \quad \text{where } \alpha' = \lambda\sigma \cdot \begin{cases} \alpha(\sigma) & \text{if } b \text{ is } \mathit{true} \text{ in state } \sigma \\ \emptyset & \text{if } b \text{ is } \mathit{false} \text{ in state } \sigma \end{cases}$$

gives rise to a label α' which is in general not a state transformation, since α' may map a state to the empty set (see Conclusion).

Instead of state transformations as labels, we could also have used pairs of states (cf. [BKPR91]). By changing the definitions of the transition relation and the state transformation semantics to be introduced below, an equivalent state transformation semantics can be obtained.

By means of the labelled transition system, the (intermediate) state transformation semantics $\mathcal{O}^*_{st}$ is defined. This state transformation semantics maps a configuration to a state transformation. This state transformation is obtained by composing the labels, i.e. state transformations, of the transition sequences starting from the configuration by means of composition and (finite) union.

Definition 2.4 The *(intermediate) state transformation semantics* $\mathcal{O}^*_{st} : \mathit{Conf} \to \mathit{ST}$ is defined by

$$\mathcal{O}^*_{st}(c) = \bigcup \{\, \alpha_1 \bullet \cdots \bullet \alpha_{k+1} \mid c \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E} \,\}.$$

To prove the well-definedness of the state transformation semantics $\mathcal{O}^*_{st}$, i.e. showing that composing the labels gives rise to a state transformation, we introduce a complexity measure.

Definition 2.5 The *complexity measure* $cm : (\mathit{Conf} \cup \{\mathrm{E}\}) \to I\!N$ is defined by

$$\begin{array}{ll}
cm(a, n) & = 1 \\
cm(s_1 \,;\, s_2, n) & = cm(s_1, n) + cm(s_2, n) + 1 \\
cm(s_1 + s_2, n) & = cm(s_1, n) + cm(s_2, n) + 1 \\
cm(s_1 \parallel s_2, n) & = cm(s_1, n) + cm(s_2, n) + 1 \\
cm([s], n) & = cm(s, n) + 2 \\
cm(x, n) & = \begin{cases} 1 & \text{if } n = 0 \\ cm(d(x), n-1) & \text{otherwise} \end{cases} \\
cm(c_1 \,;\, c_2) & = cm(c_1) + cm(c_2) \\
cm(c_1 + c_2) & = cm(c_1) + cm(c_2) \\
cm(c_1 \parallel c_2) & = cm(c_1) + cm(c_2) \\
cm([c]) & = cm(c) + 1 \\
cm(\mathrm{E}) & = 0
\end{array}$$

The complexity measure can be shown to be well-defined as follows. First, for all s and n, the well-definedness of $cm\,(s, n)$ is proved by induction on n and structural induction on s. Second, for all c, $cm\,(c)$ is demonstrated to be well-defined by structural induction on c. The complexity measure is such that if there is a transition from configuration c to configuration $\bar{c}$ (including the empty statement), then the complexity of c is greater (with respect to the lexicographic order) than that of $\bar{c}$, as is shown in

PROPERTY 2.6 *For all c and $\bar{c}$, if $c \xrightarrow{\alpha} \bar{c}$ for some α, then $cm\,(c) > cm\,(\bar{c})$.*

PROOF This property can be proved by induction on the complexity of configuration c. □

The labelled transition system is finitely branching, as is shown in

PROPERTY 2.7 *For all c, the set $\{\, (\alpha, \bar{c}) \mid c \xrightarrow{\alpha} \bar{c} \,\}$ is nonempty and finite.*

PROOF The property is proved by induction on the complexity of configuration c. We only consider the case $c \equiv [c']$. According to the definition of the transition relation,

$$\{\, (\alpha, \bar{c}) \mid [c'] \xrightarrow{\alpha} \bar{c} \,\} = \{\, (\alpha_1 \bullet \cdots \bullet \alpha_{k+1}, \mathrm{E}) \mid c' \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E} \,\}. \tag{2.1}$$

Consider the tree of which the nodes are labelled with configurations and the root is labelled with c'. The branches of the tree are labelled with state transformations, and there exists a branch from $\tilde{c}$ to $\tilde{c}'$ labelled with α if and only if $\tilde{c} \xrightarrow{\alpha} \tilde{c}'$. The paths of this tree correspond to the transition sequences started from configuration c'. Because $I\!N$ is well-founded and by Property 2.6, all transition sequences are finite, i.e. all paths in the tree are finite. By induction, the tree is finitely branching. By König's lemma ([Kön26]), the tree has only a finite number of paths. Consequently, the set (2.1) is finite. Obviously, (2.1) is a nonempty set. □

The well-definedness of the state transformation semantics $\mathcal{O}^*_{st}$ is concluded from the above two properties in

PROPERTY 2.8 *$\mathcal{O}^*_{st}$ is well-defined.*

PROOF By Property 2.7, each transition sequence is nonempty. By Property 2.6, all transition sequences are finite. Similar to the proof of Property 2.7, the nonemptiness and finiteness of the set of transition sequences started in a fixed configuration can be proved. Because composition of a nonempty and finite sequence of state transformations gives rise to a state transformation and union of a nonempty and finite set of state transformations gives rise to a state transformation, $\mathcal{O}^*_{st}$ assigns to each configuration a state transformation. □

By means of the above defined state transformation semantics $\mathcal{O}^*_{st}$, the state transformation semantics $\mathcal{O}_{st}$, which maps statements to state transformations, is defined.

DEFINITION 2.9 The *state transformation semantics* $\mathcal{O}_{st} : \mathit{Stat} \to \mathit{ST}$ is defined by

$$\mathcal{O}_{st}\,(s) = \bigsqcup_n \mathcal{O}^*_{st}\,(s, n).$$

REMARK 2.10 The above definition shows some similarities with the approximation theorem in the labelled λ-calculus, i.e.

$$[\![M]\!] = \bigsqcup \{ [\![L]\!] \mid L \in \omega(M) \},$$

with $\omega(M)$ the set of so-called Ω-approximants of the term M, as is found in [Hyl76].

The least upper bound $\bigsqcup$ in Definition 2.9 exists, because ST is a complete partial order (Property 1.2), and $(\mathcal{O}^*_{st}(s,n))_n$ is an increasing chain as is shown in

PROPERTY 2.11 *For all n, $\mathcal{O}^*_{st}(s,n) \sqsubseteq \mathcal{O}^*_{st}(s,n+1)$.*

PROOF If there exists a transition sequence $(s,n) \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E}$ with $\alpha_j = \lambda\sigma \cdot \Sigma_\perp$ for some j, then $\mathcal{O}^*_{st}(s,n) = \lambda\sigma \cdot \Sigma_\perp$, since $\bullet$ and $\cup$ are strict (Property 1.4). Consequently, $\mathcal{O}^*_{st}(s,n) \sqsubseteq \mathcal{O}^*_{st}(s,n+1)$. Otherwise, $\mathcal{O}^*_{st}(s,n) = \mathcal{O}^*_{st}(s,n+1)$. $\Box$

2.2 PREDICATE TRANSFORMER SEMANTICS

Second, we present the predicate transformer semantics. Also this semantics is defined by means of a labelled transition system. The configurations of the labelled transition system are defined as in the previous subsection. The labels of the labelled transition system are predicate transformers. With each atomic action a, a predicate transformer $\mathcal{PT}(a)$ is associated, by defining $\mathcal{PT}(a) = \omega(\mathcal{ST}(a))$. The transition relation of the labelled transition system is defined in

DEFINITION 2.12 The transition relation $\rightarrow$ is the smallest subset of

$$(Conf \cup \{\mathrm{E}\}) \times PT \times (Conf \cup \{\mathrm{E}\})$$

satisfying

- $(a,n) \xrightarrow{\mathcal{PT}(a)} \mathrm{E}$
- $(x,0) \xrightarrow{\lambda S \cdot \emptyset} \mathrm{E}$
- $\dfrac{(s_1,n)\,;(s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}{(s_1\,;s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}$
- $\dfrac{(s_1,n)+(s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}{(s_1+s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}$
- $\dfrac{(s_1,n) \parallel (s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}{(s_1 \parallel s_2,n) \xrightarrow{\beta} \mathrm{E} \mid c}$
- $\dfrac{[(s,n)] \xrightarrow{\beta} \mathrm{E} \mid c}{([s],n) \xrightarrow{\beta} \mathrm{E} \mid c}$
- $\dfrac{(d(x),n) \xrightarrow{\beta} \mathrm{E} \mid c}{(x,n+1) \xrightarrow{\beta} \mathrm{E} \mid c}$
- $\dfrac{c_2 \xrightarrow{\beta} \mathrm{E} \mid c_2'}{c_1\,;c_2 \xrightarrow{\beta} c_1 \mid c_1\,;c_2'}$
- $\dfrac{c \xrightarrow{\beta_1} c_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_{k+1}} \mathrm{E}}{[c] \xrightarrow{\beta_1 \bullet \cdots \bullet \beta_{k+1}} \mathrm{E}}$

- $$\frac{c_1 \xrightarrow{\beta} \mathrm{E} \mid c_1'}{\begin{array}{l} c_1 + c_2 \xrightarrow{\beta} \mathrm{E} \mid c_1' \\ c_2 + c_1 \xrightarrow{\beta} \mathrm{E} \mid c_1' \end{array}}$$

- $$\frac{c_1 \xrightarrow{\beta} \mathrm{E} \mid c_1'}{\begin{array}{l} c_1 \parallel c_2 \xrightarrow{\beta} c_2 \mid c_1' \parallel c_2 \\ c_2 \parallel c_1 \xrightarrow{\beta} c_2 \mid c_2 \parallel c_1' \end{array}}$$

The main difference with the transition relation defined in the previous subsection is the transition rule for the sequential composition. The configuration $c_1 \,; c_2$ can do a β-step if and only if c_2 can do a β-step (order reversal). Furthermore, the axioms for atomic actions and procedure variables and the rule for atomisation exhibit the natural differences in that

- for the atomic action a we use the label $\mathcal{PT}\,(a)$ instead of $\mathcal{ST}\,(a)$,
- for the configuration $(x, 0)$ we employ the label $\lambda S \cdot \emptyset$ instead of $\lambda\sigma \cdot \Sigma_\perp$ (note that $\omega\,(\lambda\sigma \cdot \Sigma_\perp) = \lambda S \cdot \emptyset$), and
- in the rule for atomisation we apply the composition $\bullet$ on predicate transformers instead of state transformations.

By means of the labelled transition system, the (intermediate) predicate transformer semantics $\mathcal{O}^*_{pt}$ is defined. This predicate transformer semantics maps a configuration to a predicate transformer. This predicate transformer is obtained by composing the labels, i.e. predicate transformers, of the transition sequences starting from the configuration by means of composition and (finite) intersection.

Definition 2.13 The *(intermediate) predicate transformer semantics* $\mathcal{O}^*_{pt} : \mathit{Conf} \to \mathit{PT}$ is defined by

$$\mathcal{O}^*_{pt}\,(c) = \bigcap \{\, \beta_1 \bullet \cdots \bullet \beta_{k+1} \mid c \xrightarrow{\beta_1} c_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_{k+1}} \mathrm{E} \,\}.$$

The well-definedness of the predicate transformer semantics $\mathcal{O}^*_{pt}$ can be proved along the lines of the well-definedness proof of the state transformation semantics $\mathcal{O}^*_{st}$ in the previous subsection.

Remark 2.14 The above definition shows some similarities with the definition of the weakest invariant in terms of the weakest liberal precondition, i.e.

$$\mathit{win}\,(\sigma, Q) = \bigwedge_{\lambda \in \sigma^*} \mathit{wlp}\,(\lambda, Q),$$

as at page 408 of [Lam90].

By means of the above defined predicate transformer semantics $\mathcal{O}^*_{pt}$, the predicate transformer semantics $\mathcal{O}_{pt}$, which maps statements to predicate transformers, is defined.

Definition 2.15 The *predicate transformer semantics* $\mathcal{O}_{pt} : \mathit{Stat} \to \mathit{PT}$ is defined by

$$\mathcal{O}_{pt}\,(s) = \bigsqcup_n \mathcal{O}^*_{pt}\,(s, n).$$

2.3 Isomorphism theorem

Third, we prove the main theorem of our paper establishing the state transformation semantics $\mathcal{O}_{st}$ and the predicate transformer semantics $\mathcal{O}_{pt}$ being isomorphic. As already mentioned in the introduction, a problem in the proof of this theorem is that the semantics $\mathcal{O}_{st}$ and $\mathcal{O}_{pt}$ are not compositional. As an intermediate tool, we introduce the semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$. These semantics will turn out to be compositional. The compositionality of these semantic models will facilitate the subsequent proof that they are isomorphic. From this isomorphism result we will obtain the isomorphism of $\mathcal{O}_{st}$ and $\mathcal{O}_{pt}$.

The semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are defined by means of the labelled transition systems defining the intermediate semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{st}$, respectively. They map a configuration to a set of *sequences* of state transformations and predicate transformers. To each configuration the set of label sequences corresponding to the transition sequences starting from the configuration is assigned. The sets of nonempty and finite sequences of state transformations and predicate transformers are denoted by ST^+ and PT^+, and the sets of nonempty and finite subsets of these sets are denoted by $\mathcal{P}_{nf}(ST^+)$ and $\mathcal{P}_{nf}(PT^+)$.

Definition 2.16 The semantics $\mathcal{C}_{st} : \mathit{Conf} \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$\mathcal{C}_{st}(c) = \{ \alpha_1 \cdots \alpha_{k+1} \mid c \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E} \}$$

and the semantics $\mathcal{C}_{pt} : \mathit{Conf} \to \mathcal{P}_{nf}(PT^+)$ is defined by

$$\mathcal{C}_{pt}(c) = \{ \beta_1 \cdots \beta_{k+1} \mid c \xrightarrow{\beta_1} c_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_{k+1}} \mathrm{E} \}.$$

The well-definedness of these semantics can be proved along the lines of the well-definedness proofs of the intermediate semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{pt}$. We now show that the above introduced semantics are compositional by introducing for each syntactic operator a corresponding semantic operator (denoted by the same symbol).

Definition 2.17 The mapping $; : \mathcal{P}_{nf}(ST^+) \times \mathcal{P}_{nf}(ST^+) \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$X \,;\, X' = \{ x \,;\, x' \mid x \in X \wedge x' \in X' \}$$

where

$$\begin{aligned} \alpha \,;\, x' &= \alpha x' \\ \alpha x \,;\, x' &= \alpha(x \,;\, x') \end{aligned}$$

The mapping $+ : \mathcal{P}_{nf}(ST^+) \times \mathcal{P}_{nf}(ST^+) \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$X + X' = X \cup X'.$$

The mapping $\| : \mathcal{P}_{nf}(ST^+) \times \mathcal{P}_{nf}(ST^+) \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$X \parallel X' = \bigcup \{ x \mathbin{\lfloor\!\lfloor} x' + x' \mathbin{\lfloor\!\lfloor} x \mid x \in X \wedge x' \in X' \}$$

where

$$\begin{aligned} \alpha \mathbin{\lfloor\!\lfloor} x' &= \{\alpha x'\} \\ \alpha x \mathbin{\lfloor\!\lfloor} x' &= \alpha(x \parallel x') \end{aligned}$$

The mapping $[\,]: \mathcal{P}_{nf}(ST^+) \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$[X] = \{\, [x] \mid x \in X \,\}$$

where

$$\begin{aligned} [\alpha] &= \alpha \\ [\alpha x] &= \alpha \bullet [x] \end{aligned}$$

The compositionality of $\mathcal{C}_{st}$ is shown in

PROPERTY 2.18 *For the semantics $\mathcal{C}_{st}$ we have that*

$$\begin{aligned}
\mathcal{C}_{st}(a, n) &= \{\mathcal{ST}(a)\} \\
\mathcal{C}_{st}(s_1 \,;\, s_2, n) &= \mathcal{C}_{st}(s_1, n) \,;\, \mathcal{C}_{st}(s_2, n) \\
\mathcal{C}_{st}(s_1 + s_2, n) &= \mathcal{C}_{st}(s_1, n) + \mathcal{C}_{st}(s_2, n) \\
\mathcal{C}_{st}(s_1 \parallel s_2, n) &= \mathcal{C}_{st}(s_1, n) \parallel \mathcal{C}_{st}(s_2, n) \\
\mathcal{C}_{st}([s], n) &= [\mathcal{C}_{st}(s, n)] \\
\mathcal{C}_{st}(x, 0) &= \{\lambda\sigma \cdot \Sigma_\perp\} \\
\mathcal{C}_{st}(x, n+1) &= \mathcal{C}_{st}(d(x), n) \\
\mathcal{C}_{st}(c_1 \,;\, c_2) &= \mathcal{C}_{st}(c_1) \,;\, \mathcal{C}_{st}(c_2) \\
\mathcal{C}_{st}(c_1 + c_2) &= \mathcal{C}_{st}(c_1) + \mathcal{C}_{st}(c_2) \\
\mathcal{C}_{st}(c_1 \parallel c_2) &= \mathcal{C}_{st}(c_1) \parallel \mathcal{C}_{st}(c_2) \\
\mathcal{C}_{st}([c]) &= [\mathcal{C}_{st}(c)]
\end{aligned}$$

PROOF The property is proved by induction on the complexity of the configuration. Below, the notation

$$\begin{aligned} \alpha X &= \{\, \alpha x \mid x \in X \,\} \\ \alpha \bullet X &= \{\, \alpha \bullet x \mid x \in X \,\} \end{aligned}$$

is used. Only a few cases are elaborated on.

1. Let $c \equiv (s_1 \parallel s_2, n)$.

$$\begin{aligned} &\mathcal{C}_{st}(s_1 \parallel s_2, n) \\ &= \mathcal{C}_{st}((s_1, n) \parallel (s_2, n)) \\ &= \mathcal{C}_{st}(s_1, n) \parallel \mathcal{C}_{st}(s_2, n). \quad \text{[induction]} \end{aligned}$$

2. Let $c \equiv c_1 \,;\, c_2$.

$$\begin{aligned} &\mathcal{C}_{st}(c_1 \,;\, c_2) \\ &= \bigcup \{\, \alpha\ \mathcal{C}_{st}(c_2) \mid c_1 \xrightarrow{\alpha} \mathrm{E} \,\} \cup \bigcup \{\, \alpha\ \mathcal{C}_{st}(c_1' \,;\, c_2) \mid c_1 \xrightarrow{\alpha} c_1' \,\} \\ &= \bigcup \{\, \alpha\ \mathcal{C}_{st}(c_2) \mid c_1 \xrightarrow{\alpha} \mathrm{E} \,\} \cup \bigcup \{\, \alpha\ (\mathcal{C}_{st}(c_1') \,;\, \mathcal{C}_{st}(c_2)) \mid c_1 \xrightarrow{\alpha} c_1' \,\} \quad \text{[Property 2.6, induction]} \\ &= (\{\, \alpha \mid c_1 \xrightarrow{\alpha} \mathrm{E} \,\} \cup \bigcup \{\, \alpha\ \mathcal{C}_{st}(c_1') \mid c_1 \xrightarrow{\alpha} c_1' \,\}) \,;\, \mathcal{C}_{st}(c_2) \\ &= \mathcal{C}_{st}(c_1) \,;\, \mathcal{C}_{st}(c_2). \end{aligned}$$

3. Let $c \equiv [c']$.

$$\begin{aligned} &\mathcal{C}_{st}([c']) \\ &= \{\, \alpha_1 \bullet \cdots \bullet \alpha_{k+1} \mid c' \xrightarrow{\alpha_1} c_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{k+1}} \mathrm{E} \,\} \end{aligned}$$

$$
\begin{array}{ll}
= & \{\, \alpha_1 \mid c' \xrightarrow{\alpha_1} \mathrm{E} \,\} \cup \bigcup \{\, \alpha_1 \bullet \mathcal{C}_{st}([c_1]) \mid c' \xrightarrow{\alpha_1} c_1 \,\} \\
= & \{\, \alpha_1 \mid c' \xrightarrow{\alpha_1} \mathrm{E} \,\} \cup \bigcup \{\, \alpha_1 \bullet [\mathcal{C}_{st}(c_1)] \mid c' \xrightarrow{\alpha_1} c_1 \,\} \quad \text{[Property 2.6, induction]} \\
= & [\{\, \alpha_1 \mid c' \xrightarrow{\alpha_1} \mathrm{E} \,\} \cup \bigcup \{\, \alpha_1 \; \mathcal{C}_{st}(c_1) \mid c' \xrightarrow{\alpha_1} c_1 \,\}] \\
= & [\mathcal{C}_{st}(c')].
\end{array}
$$

□

In order to show that $\mathcal{C}_{pt}$ is compositional, semantic operators similar to the ones introduced in Definition 2.17 can be introduced. The compositionality of $\mathcal{C}_{pt}$ is demonstrated in

PROPERTY 2.19 *For the semantics $\mathcal{C}_{pt}$ we have that*

$$
\begin{array}{ll}
\mathcal{C}_{pt}(a, n) & = \{\mathcal{PT}(a)\} \\
\mathcal{C}_{pt}(s_1 \,;\, s_2, n) & = \mathcal{C}_{pt}(s_2, n) \,;\, \mathcal{C}_{pt}(s_1, n) \\
\mathcal{C}_{pt}(s_1 + s_2, n) & = \mathcal{C}_{pt}(s_1, n) + \mathcal{C}_{pt}(s_2, n) \\
\mathcal{C}_{pt}(s_1 \parallel s_2, n) & = \mathcal{C}_{pt}(s_1, n) \parallel \mathcal{C}_{pt}(s_2, n) \\
\mathcal{C}_{pt}([s], n) & = [\mathcal{C}_{pt}(s, n)] \\
\mathcal{C}_{pt}(x, 0) & = \{\lambda S \cdot \emptyset\} \\
\mathcal{C}_{pt}(x, n+1) & = \mathcal{C}_{pt}(d(x), n) \\
\mathcal{C}_{pt}(c_1 \,;\, c_2) & = \mathcal{C}_{pt}(c_2) \,;\, \mathcal{C}_{pt}(c_1) \\
\mathcal{C}_{pt}(c_1 + c_2) & = \mathcal{C}_{pt}(c_1) + \mathcal{C}_{pt}(c_2) \\
\mathcal{C}_{pt}(c_1 \parallel c_2) & = \mathcal{C}_{pt}(c_1) \parallel \mathcal{C}_{pt}(c_2) \\
\mathcal{C}_{pt}([c]) & = [\mathcal{C}_{pt}(c)]
\end{array}
$$

PROOF Similar to the proof of Property 2.18. □

The main difference with the previous property is the clause for the sequential composition. The semantics of the sequential composition of c_1 and c_2 is the sequential composition of the semantics of c_2 and the semantics of c_1 (order reversal).

Next, we show that the compositional semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are isomorphic. For this purpose, we first prove their codomains, i.e. $\mathcal{P}_{nf}(ST^+)$ and $\mathcal{P}_{nf}(PT^+)$, to be isomorphic. The isomorphism Ω reverses each sequence of state transformations and applies ω (cf. Definition 1.10) to each state transformation of the reversed sequence. Its inverse Ω^{-1} reverses each sequence of predicate transformers and applies ω^{-1} to each predicate transformer of the reversed sequence.

DEFINITION 2.20 The mapping $\Omega : \mathcal{P}_{nf}(ST^+) \to \mathcal{P}_{nf}(PT^+)$ is defined by

$$\Omega(X) = \{\, \omega(\alpha_{k+1}) \cdots \omega(\alpha_1) \mid \alpha_1 \cdots \alpha_{k+1} \in X \,\}$$

and the mapping $\Omega^{-1} : \mathcal{P}_{nf}(PT^+) \to \mathcal{P}_{nf}(ST^+)$ is defined by

$$\Omega^{-1}(Y) = \{\, \omega^{-1}(\beta_{k+1}) \cdots \omega^{-1}(\beta_1) \mid \beta_1 \cdots \beta_{k+1} \in Y \,\}.$$

PROPERTY 2.21 *The mapping $\Omega : \mathcal{P}_{nf}(ST^+) \cong \mathcal{P}_{nf}(PT^+)$ is an isomorphism of sets.*

PROOF Immediate consequence of Theorem 1.11. □

By means of this isomorphism, we can prove the compositional semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ to be isomorphic.

$$\begin{tikzcd}
 & Conf \arrow[dl, "\mathcal{C}_{st}"'] \arrow[dr, "\mathcal{C}_{pt}"] \arrow[d, phantom, "*"] & \\
\mathcal{P}_{nf}\,(ST^{+}) \arrow[rr, leftrightarrow, "\Omega"'] & & \mathcal{P}_{nf}\,(PT^{+})
\end{tikzcd}$$

The isomorphism of $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ is based on the following properties of the isomorphism Ω.

Property 2.22 *For all X and X',*

$$\begin{aligned}
\Omega\,(X\,;X') &= \Omega\,(X')\,;\Omega\,(X)\\
\Omega\,(X+X') &= \Omega\,(X)+\Omega\,(X')\\
\Omega\,(X\parallel X') &= \Omega\,(X)\parallel\Omega\,(X')\\
\Omega\,([X]) &= [\Omega\,(X)]
\end{aligned}$$

Proof Only the last case is considered.

$$\begin{aligned}
&\Omega\,([X])\\
&\quad=\quad \Omega\,(\{\,\alpha_1\bullet\cdots\bullet\alpha_{k+1}\mid\alpha_1\cdots\alpha_{k+1}\in X\,\})\\
&\quad=\quad \{\,\omega\,(\alpha_1\bullet\cdots\bullet\alpha_{k+1})\mid\alpha_1\cdots\alpha_{k+1}\in X\,\}\\
&\quad=\quad \{\,\omega\,(\alpha_{k+1})\bullet\cdots\bullet\omega\,(\alpha_1)\mid\alpha_1\cdots\alpha_{k+1}\in X\,\}\quad[\text{Property 1.12}]\\
&\quad=\quad [\{\,\omega\,(\alpha_{k+1})\cdots\omega\,(\alpha_1)\mid\alpha_1\cdots\alpha_{k+1}\in X\,\}]\\
&\quad=\quad [\Omega\,(X)].
\end{aligned}$$

□

Note that the Ω-image of the sequential composition of X and X' is the sequential composition of the Ω-image of X' and the Ω-image of X (order reversal). For the other operators, Ω is a homomorphism.

Property 2.23 *$\Omega\circ\mathcal{C}_{st}=\mathcal{C}_{pt}$ and $\Omega^{-1}\circ\mathcal{C}_{pt}=\mathcal{C}_{st}$.*

Proof We prove the property by induction on the complexity of configuration c. Only the case $c\equiv c_1\,;c_2$ is considered.

$$\begin{aligned}
&\Omega\,(\mathcal{C}_{st}\,(c_1\,;c_2))\\
&\quad=\quad \Omega\,(\mathcal{C}_{st}\,(c_1)\,;\mathcal{C}_{st}(c_2)) && [\text{Property 2.18}]\\
&\quad=\quad \Omega\,(\mathcal{C}_{st}\,(c_2))\,;\Omega\,(\mathcal{C}_{st}\,(c_1)) && [\text{Property 2.22}]\\
&\quad=\quad \mathcal{C}_{pt}\,(c_2)\,;\mathcal{C}_{pt}\,(c_1) && [\text{induction}]\\
&\quad=\quad \mathcal{C}_{pt}\,(c_1\,;c_2). && [\text{Property 2.19}]
\end{aligned}$$

Furthermore,

$$\begin{aligned}
&\Omega^{-1}\,(\mathcal{C}_{pt}\,(c))\\
&\quad=\quad \Omega^{-1}\,(\Omega\,(\mathcal{C}_{st}\,(c)))\\
&\quad=\quad \mathcal{C}_{st}\,(c). && [\text{Property 2.21}]
\end{aligned}$$

□

We introduce abstraction operators abs_{st} and abs_{pt}. These operators composing sets of label sequences as is done in the definitions of the intermediate semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{pt}$, are defined in

Definition 2.24 The mapping $abs_{st} : \mathcal{P}_{nf}(ST^+) \to ST$ is defined by

$$abs_{st}(X) = \bigcup \{ \alpha_1 \bullet \cdots \bullet \alpha_{k+1} \mid \alpha_1 \cdots \alpha_{k+1} \in X \}$$

and the mapping $abs_{pt} : \mathcal{P}_{nf}(PT^+) \to PT$ is defined by

$$abs_{pt}(Y) = \bigcap \{ \beta_1 \bullet \cdots \bullet \beta_{k+1} \mid \beta_1 \cdots \beta_{k+1} \in Y \}.$$

The isomorphisms ω and Ω are related by means of the abstraction operators.

$$\begin{tikzcd}
\mathcal{P}_{nf}(ST^+) \arrow[r, leftrightarrow, "\Omega"] \arrow[d, "abs_{st}"'] \arrow[dr, phantom, "*"] & \mathcal{P}_{nf}(PT^+) \arrow[d, "abs_{pt}"] \\
ST \arrow[r, leftrightarrow, "\omega"'] & PT
\end{tikzcd}$$

Property 2.25 $\omega \circ abs_{st} = abs_{pt} \circ \Omega$.

Proof

$$\begin{array}{lll}
\multicolumn{3}{l}{\omega(abs_{st}(X))} \\
= & \omega(\bigcup \{ \alpha_1 \bullet \cdots \bullet \alpha_{k+1} \mid \alpha_1 \cdots \alpha_{k+1} \in X \}) & \\
= & \bigcap \{ \omega(\alpha_1 \bullet \cdots \bullet \alpha_{k+1}) \mid \alpha_1 \cdots \alpha_{k+1} \in X \} & \text{[Property 1.12]} \\
= & \bigcap \{ \omega(\alpha_{k+1}) \bullet \cdots \bullet \omega(\alpha_1) \mid \alpha_1 \cdots \alpha_{k+1} \in X \} & \text{[Property 1.12]} \\
= & abs_{pt}(\{ \omega(\alpha_{k+1}) \cdots \omega(\alpha_1) \mid \alpha_1 \cdots \alpha_{k+1} \in X \}) & \\
= & abs_{pt}(\Omega(X)). &
\end{array}$$

□

The compositional semantics $\mathcal{C}_{st}$ and $\mathcal{C}_{pt}$ are related to the intermediate semantics $\mathcal{O}^*_{st}$ and $\mathcal{O}^*_{pt}$ by means of the abstraction operators.

$$\begin{tikzcd}
\mathcal{P}_{nf}(ST^+) \arrow[dd, "abs_{st}"'] & & & & \mathcal{P}_{nf}(PT^+) \arrow[dd, "abs_{pt}"] \\
& * & Conf \arrow[ull, "\mathcal{C}_{st}"'] \arrow[urr, "\mathcal{C}_{pt}"] \arrow[dll, "\mathcal{O}^*_{st}"] \arrow[drr, "\mathcal{O}^*_{pt}"'] & * & \\
ST & & & & PT
\end{tikzcd}$$

Property 2.26 $abs_{st} \circ \mathcal{C}_{st} = \mathcal{O}^*_{st}$ *and* $abs_{pt} \circ \mathcal{C}_{pt} = \mathcal{O}^*_{pt}$.

Proof Trivial. □

Finally, we prove that the state transformation semantics $\mathcal{O}_{st}$ and the predicate transformer semantics $\mathcal{O}_{pt}$ are isomorphic by combining the Properties 2.23, 2.25, and 2.26.

$$\begin{tikzcd} & Stat \arrow[dl, "\mathcal{O}_{st}"'] \arrow[dr, "\mathcal{O}_{pt}"] \arrow[d, phantom, "*"] & \\ ST \arrow[rr, leftrightarrow, "\omega"'] & & PT \end{tikzcd}$$

THEOREM 2.27 *$\omega \circ \mathcal{O}_{st} = \mathcal{O}_{pt}$ and $\omega^{-1} \circ \mathcal{O}_{pt} = \mathcal{O}_{st}$.*

PROOF

$$\begin{array}{lll} \multicolumn{3}{l}{\omega(\mathcal{O}_{st}(s))} \\ = & \omega(\bigsqcup_n \mathcal{O}^*_{st}(s,n)) & \\ = & \bigsqcup_n \omega(\mathcal{O}^*_{st}(s,n)) & \text{[Theorem 1.11]} \\ = & \bigsqcup_n \omega(abs_{st}(\mathcal{C}_{st}(s,n))) & \text{[Property 2.26]} \\ = & \bigsqcup_n abs_{pt}(\Omega(\mathcal{C}_{st}(s,n))) & \text{[Property 2.25]} \\ = & \bigsqcup_n abs_{pt}(\mathcal{C}_{pt}(s,n)) & \text{[Property 2.23]} \\ = & \bigsqcup_n \mathcal{O}^*_{pt}(s,n) & \text{[Property 2.26]} \\ = & \mathcal{O}_{pt}(s). & \end{array}$$

Furthermore,

$$\begin{array}{lll} \multicolumn{3}{l}{\omega^{-1}(\mathcal{O}_{pt}(s))} \\ = & \omega^{-1}(\omega(\mathcal{O}_{st}(s))) & \\ = & \mathcal{O}_{st}(s). & \text{[Theorem 1.11]} \end{array}$$

□

CONCLUSION

A state transformation semantics and a predicate transformer semantics for programs built from atomic actions, sequential composition, nondeterministic choice, parallel composition, atomisation, and recursion have been presented. These semantics were shown to be isomorphic. Although parallel composition fits in the presented framework, for example, the conditional statement does not fit in the framework (see Section 2).

In order to treat the conditional statement, we could drop the restriction to nonempty sets in Definition 1.1. The modified complete partial order of state transformations is isomorphic to the complete partial order of continuous and multiplicative predicate transformers (cf. [BK92]). By means of this isomorphism we could extend the presented results and deal also with the conditional statement.

The framework presented in our paper might also be amended to treat a variety of other language constructs using the various isomorphism results mentioned in the introduction.

References

[AP86] K.R. Apt and G.D. Plotkin. Countable Nondeterminism and Random Assignment. *Journal of the ACM*, 33(4):724–767, October 1986.

[Bak77] J.W. de Bakker. Recursive Programs as Predicate Transformers. In E.J. Neuhold, editor, *Proceedings of IFIP Working Conference on Formal Description of Programming Concepts*, pages 165–179, St. Andrews, August 1977. North-Holland Publishing Company.

[Bak80] J.W. de Bakker. *Mathematical Theory of Program Correctness.* Series in Computer Science. Prentice-Hall International, 1980.

[Ber78] G. Berry. Stable Models of Typed λ-Calculi. In G. Ausiello and C. Böhm, editors, *Proceedings of 5th International Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 72–89, Udine, July 1978. Springer-Verlag.

[Bes82] E. Best. Relational Semantics of Concurrent Programs. In D. Bjørner, editor, *Proceedings of IFIP Working Conference on Formal Description of Programming Concepts - II*, pages 431–452, Garmisch-Partenkirchen, June 1982. North-Holland Publishing Company.

[Bes89] E. Best. Towards Compositional Predicate Transformer Semantics for Concurrent Programs. In *J.W. de Bakker, 25 jaar semantiek*, pages 111–117. CWI, Amsterdam, April 1989.

[BK92] M. Bonsangue and J.N. Kok. Semantics, Orderings and Recursion in the Weakest Precondition Calculus. Report CS-R9267, CWI, Amsterdam, December 1992. Extended abstract appeared in J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop on Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 91–109, Beekbergen, June 1992. Springer-Verlag.

[BK93] M. Bonsangue and J.N. Kok. Isomorphisms between Predicate and State Transformers. To appear in *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, *Lecture Notes in Computer Science*, Gdansk, August/September 1993. Springer-Verlag.

[BKPR91] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. The Failure of Failures in a Paradigm for Asynchronous Communication. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 111–126, Amsterdam, August 1991. Springer-Verlag.

[Coo78] S.A. Cook. Soundness and Completeness of an Axiom System for Program Verification. *SIAM Journal of Computation*, 7(1):70–90, February 1978.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Series in Automatic Computation. Prentice-Hall International, 1976.

[EF84] T. Elrad and N. Francez. A Weakest Precondition Semantics for Communicating Processes. *Theoretical Computer Science*, 29(3):231–250, 1984.

[FS81] L. Flon and N. Suzuki. The Total Correctness of Parallel Programs. *SIAM Journal of Computation*, 10(2):227–246, May 1981.

[Haa81] V.H. Haase. Real-Time Behaviour of Programs. *IEEE Transactions on Software Engineering*, SE-7(5):494–501, September 1981.

[Hyl76] M. Hyland. A Syntactic Characterization of the Equality of some Models for the Lambda Calculus. *Journal of the London Mathematical Association*, 12(2):361–370, 1976.

[Kön26] D. König. Sur les correspondences multivoques des emsembles. *Fundamenta Mathematicae*, 8:114–134, 1926.

[Lam90] L. Lamport. *win* and *sin*: Predicate Transformers for Concurrency. *ACM Transactions on Programming Languages and Systems*, 12(3):396–428, July 1990.

[LL90] L. Lamport and N. Lynch. Distributed Computing: Models and Methods. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 18, pages 1157–1199. The MIT Press/Elsevier, Cambridge/Amsterdam, 1990.

[LS79] L. van Lamsweerde and M. Sintzoff. Formal Derivation of Strongly Correct Concurrent Programs. *Acta Informatica*, 12(1):1–31, 1979.

[MC80] M.E. Majster-Cederbaum. A Simple Relation between Relational and Predicate Transformer Semantics for Nondeterministic Programs. *Information Processing Letters*, 11(4/5):190–192, December 1980.

[Mil93] R. Milner. Elements of Interaction. *Communications of the ACM*, 36(1):78–89, January 1993.

[Plo79] G.D. Plotkin. Dijkstra's Predicate Transformers and Smyth's Powerdomains. In D. Bjørner, editor, *Proceedings of the Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 527–553, Copenhagen, January/February 1979. Springer-Verlag.

[Plo81] G.D. Plotkin. A Structural Approach to Operational Semantics. Report DAIMI FN-19, Aarhus University, Aarhus, September 1981.

[Plo82] G.D. Plotkin. An Operational Semantics for CSP. In D. Bjørner, editor, *Proceedings of IFIP Working Conference on Formal Description of Programming Concepts - II*, pages 199–223, Garmisch-Partenkirchen, June 1982. North-Holland Publishing Company.

[Roe76] W.-P. de Roever. Dijkstra's Predicate Transformer, Non-Determinism, Recursion and Termination. In A. Mazurkiewicz, editor, *Proceedings of 5th Symposium on Mathematical Foundations of Computer Science*, volume 45 of *Lecture Notes in Computer Science*, pages 472–481, Gdansk, September 1976. Springer-Verlag.

[Smy78] M.B. Smyth. Power Domains. *Journal of Computer and System Sciences*, 16(1):23–36, February 1978.

[Smy83] M.B. Smyth. Power Domains and Predicate Transformers: a Topological View. In J. Diaz, editor, *Proceedings of 10th International Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675, Barcelona, July 1983. Springer-Verlag.

[Smy92] M.B. Smyth. Topology. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, Background: Mathematical Structures, pages 641–761. Clarendon Press, Oxford, 1992.

[SZ92] D. Scholefield and H.S.M. Zedan. Weakest Precondition Semantics for Time and Concurrency. *Information Processing Letters*, 43(6):301–308, October 1992.

[Wan77] M. Wand. A Characterization of Weakest Preconditions. *Journal of Computer and System Sciences*, 15(2):209–212, October 1977.