

1992

R.T.P. Fernando

Comparative transition system semantics

Computer Science/Department of Software Technology Report CS-R9222 July

CWI is het Centrum voor Wiskunde en Informatica van de Stichting Mathematisch Centrum
CWI is the Centre for Mathematics and Computer Science of the Mathematical Centre Foundation

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

Comparative Transition System Semantics

Tim Fernando
fernando@cwi.nl

CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract. The notion of a transition system is employed to relate programs, conceived as binary (transition) relations on states, to processes, viewed as dynamic states. The comparative study is carried out syntactically through the enumeration of rules for transitions, and semantically in terms of bisimulation equivalence. A certain form of transitions is studied, and a “logical” approach to the notion of a bisimulation is taken that are somewhat non-standard (but, it is hoped, illuminating). Sequential composition, non-deterministic choice, iteration, and interleaving are analyzed alongside a notion of data. Atomization and synchronization are also considered.

1991 Mathematics Subject Classification: 03B70.

1991 Computing Reviews Categories: D.3.1.

Key words and phrases: transition system, programs as relations, processes as states, bisimulation, non-well-founded sets.

Note: This work was funded by the Netherlands Organization for Scientific Research (NWO project NF 102/62-356, ‘Structural and Semantic Parallels in Natural Languages and Programming Languages’). My thanks to Jan Willem Klop for suggesting that a report entitled “Comparative transition system semantics” be written, and for help along the way; also to Jan Rutten, Daniele Turi, Alban Ponse, Frits Vaandrager, Johan van Benthem, Prakash Panangaden, Fer-Jan de Vries, Jan van Eijck, and Franck van Breugel for useful discussions.

A (labelled) transition system is a triple $\langle L, S, \rightarrow \rangle$ consisting of a set L of labels, a set S of states, and a transition relation $\rightarrow \subseteq S \times L \times S$. This general scheme brings together various notions of “transitions” $s \xrightarrow{l} s'$ (encoded as $(s, l, s') \in \rightarrow$), where labels and states can be analyzed in terms of each other. In some studies, such as process algebra (see, for instance, Baeten and Weijland [2] and the references cited therein), a label is taken to be primitive, the main point being to analyze what a state — i.e., process — is. Another point of view reduces a label to the pairs of states that it relates via \rightarrow . Indeed, the analysis in dynamic logic (see, for example, Harel [12]) of a program as a binary relation on data-states yields a transition system whose labels are programs, and whose states are data-states. Now, although any family \mathbf{R} of binary relations on some set X can be regarded as a transition system $(\mathbf{R}, X, \rightsquigarrow)$ with

$$\rightsquigarrow = \{(x, R, y) \in X \times \mathbf{R} \times X \mid (x, y) \in R\},$$

it is not the case that every transition system can be obtained this way. A ternary relation \rightarrow can support at least two different kinds of transitions that a set of binary

Report CS-R9222
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

relations cannot (under the usual encoding of relations as sets). First, \rightarrow allows two distinct labels l and l' to relate the same states

$$\{(s, s') \mid s \xrightarrow{l} s'\} = \{(s, s') \mid s \xrightarrow{l'} s'\} .$$

Second, by the usual axioms of set theory, circular predicates such as $(l, l) \in l$ can never hold, whereas $l \xrightarrow{l} l$ might. Now, if labels are taken to be programs, then the possibility of distinct labels relating the same states is, in fact, just what the view of programs as relations on states forbids. In contrast, as will become clear below, a restriction to non-circular transitions would represent a severe limitation for an account of programs. The restriction can be relaxed, but not without introducing complications of its own, revolving around equality — complications that can be overcome either by reading “programs as relations” as “programs isomorphic to relations”, or, alternatively, by passing to the non-well-founded set theory presented in Aczel [1]. These “alternatives” turn out to be equivalent, and, what’s more, related to a well-known notion of equivalence on transition systems based on so-called bisimulations (Park [14]).

The present paper is directed towards reconciling two points of view,

(P1) a program as a (binary) relation on states

and

(P2) a process as a state in a transition system ,

enriching the former by incorporating a program into its notion of state, and the latter by passing from atomic transitions to a finite sequence of such transitions. The comparison given of (P1) and (P2) is carried out for the regular program constructs (viz., sequential composition, non-deterministic choice and iteration) and interleaving over a notion of data. Transition systems from process algebra, Groote and Ponse [10], and De Boer, Kok, Palamidessi, and Rutten [6] are presented and examined in modified forms simplified to highlight their essential characteristics (at least from the point of view of the present paper). The author expects that the analysis below can be extended to cover a wider range of constructs, including atomization and synchronization, discussed briefly in the final section and in an appendix.

The paper begins with an informal account of five transition systems from dynamic logic, process algebra and the ground in between. This is followed in section 2 by a syntactic study based on a single unifying transition system. A semantics for (P1) is worked out in section 3, providing an “alternative” approach (based on (P1)) to the notion of a bisimulation designed for (P2). Section 4 relates the different transition systems semantically by analyzing models that are in a suitable sense final. Some directions for further work are discussed in section 5, and an appendix containing certain technical details behind section 3 attached.

1 Five transition systems

The present section concentrates on essential intuitions, leaving a precise formulation of the transition systems to the next section. In addition to writing $\langle L, S, \rightarrow \rangle$ for a transition system (to be instantiated below in different ways), the following notation is adopted, and kept throughout the paper. D is a set of “data-states” d, d', d_1, \dots ; P_1 is the set of “program(term)s” p, \dots given by

$$p ::= a \mid p_1; p_2 \mid p_1 + p_2 \mid p_1 \parallel p_2 \mid p_1^*$$

over a set A of “atomic programs” a , each of which comes with an “interpretation” $I_a \subseteq D \times D$.

While there is general agreement as to what $;$, $+$ and $*$ mean, the “parallel” construct \parallel is a bit more troublesome. A view of \parallel is adopted throughout this paper, allowing the program $x := 1 \parallel (x := 0; x := x + 1)$ to end in a state where $x = 2$ (resulting from interleaving $x := 1$ between $x := 0$ and $x := x + 1$). Otherwise, the program $x := 0; x := x + 1$ should be “atomized” (see De Bakker and De Vink [3], the references cited therein, and also section 5 below) before composing it in parallel with $x := 1$. Beyond interleaving, however, a notion of synchronization is often incorporated into \parallel . Such complications are taken up in Appendix B.

1.1 Dynamic logic (semantics)

The first example is an extreme, extensional formulation of (P1) where states are inputs and/or outputs.

Example 1. Let P_0 be the set of programs in P_1 with no occurrences of \parallel . The semantic framework underlying dynamic logic provides a transition system where $L = P_0$ and $S = D$, with transitions written

$$d \llbracket p \rrbracket_0 d' \tag{1}$$

indicating that program p can yield (as output) d' , given input d (and where $\llbracket a \rrbracket_0 = I_a$). (Note: the present paper concerns only the semantic component of dynamic logic, and not the modal logic for analyzing these structures.)

A well-known fundamental limitation of (1) is that it cannot support a compositional account of parallelism in that $\llbracket x := 1 \rrbracket_0 = \llbracket x := 0; x := x + 1 \rrbracket_0$, whereas $\llbracket x := 1 \rrbracket_0 \parallel \llbracket x := 1 \rrbracket_0$ better not equal $\llbracket x := 1 \rrbracket_0 \parallel \llbracket x := 0; x := x + 1 \rrbracket_0$. The concurrent extension of dynamic logic in Peleg [15] where programs are interpreted as subsets of $D \times 2^D$ is inconsistent with this view. We consider instead approaches based on intermediate states in a computation.

1.2 Process algebra

A fine-grained, intensional analysis of programs is developed in

Example 2. Taking for granted a translation into P_1 of the corresponding processes, an example of a transition system in process algebra is provided by $L = A + \{\sqrt{}\}$ and $S = P_1 + \{\sqrt{}\}$, with transitions

$$p \xrightarrow{a} p' \quad (2)$$

indicating that process p can execute the atomic program a , and in so doing transform itself into p' . The symbol $\sqrt{}$ denotes successful termination.

1.3 Process algebra with data

Comparing Example 1 with Example 2, a question that arises is what happened to the data-states in D ? D is (re-)introduced into the picture in

Example 3. In Groote and Ponse [10], $L = A + \{\sqrt{}\}$, and $S = (P_1 + \{\sqrt{}\}) \times D$ (so-called “configurations”), with transitions of the form

$$(p, d) \xrightarrow{a}_3 (p', d') \quad (3)$$

extending Example 2 with the condition that $d I_a d'$. Under this account, it is clear that implicit in the basic axiom $a; p \xrightarrow{a} p$ for (2) is the assumption that for every “normal” $a \in A$,

$$\forall d \exists d' \ d I_a d' .$$

Atomic programs that need not have this property are central to Groote and Ponse [10], where they are called *guards*.

1.4 Data-state pairs as labels

Abstracting away the atomic program a from (3) and promoting the data-states to the center stage, above the arrow, we arrive at

Example 4. In De Boer, Kok, Palamidessi, and Rutten [6], $L = D \times D$, and $S = P_1 + \{\sqrt{}\}$, with transitions

$$p \xrightarrow{(d, d')}_4 p' \quad (4)$$

indicating that program p can turn into p' starting from an initial data-state d which is updated to d' . A program p can then be assigned denotations (following De Boer, Kok, Palamidessi, and Rutten [6]) based on sequences

$$p \xrightarrow{(d_1, d'_1)}_4 p_1 \xrightarrow{(d_2, d'_2)}_4 p_2 \dots$$

of transitions that may or may not be connected in the sense that d'_i may or may not equal d_{i+1} . (This, at least, is the case for programs without $*$ and of bounded non-determinism.)

The passage from d'_i to d_{i+1} is made explicit in the next example, where symbols in (4) are moved around a bit, and transitions caused by programs possibly more complicated than $a \in A$ are allowed.

1.5 Programs as relations on data and programs

The principle that programs are only “visible” through their effects on data is developed further in our fifth example, which trades atomicity for transitions that intuitively may take more than a single step (consistent with the absence of a global clock).

Example 5. Let $L = P_1$, $S = D \times (P_1 + \{\sqrt{}\})$, and consider transitions

$$(d_1, p_1) [p] (d_2, p_2), \quad (5)$$

reflecting the intuition that an operating system with a data-state d_1 and a suspended job p_1 can respond to an external request p by updating its data-state to d_2 and its “jobs-to-do” to p_2 . (5) reduces to (1) when $p_1 = p_2 = \sqrt{}$, and $p \in P_0$. External requests are assumed to have priority equal to internal jobs. Thus, the transitions

$$\begin{aligned} &(d, x := 1) [x := 0; x := x + 1] (d[x=1], x := 0; x := x + 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=0], x := x + 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=1], \sqrt{}) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=0], x := x + 1 \parallel x := 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=0], x := 1 \parallel x := x + 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=1], x := 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=1], x := x + 1) \\ &(d, x := 1) [x := 0; x := x + 1] (d[x=2], \sqrt{}) \end{aligned}$$

are licensed, some of which properly extend others. Similarly,

$$\begin{aligned} &(d, x := 0; x := x + 1) [x := 1] (d[x=1], x := 0; x := x + 1) \\ &\quad \vdots \\ &(d, x := 0; x := x + 1) [x := 1] (d[x=2], \sqrt{}), \end{aligned}$$

or returning to the example of $x := 1 \parallel (x := 0; x := x + 1)$,

$$\begin{aligned} &(d, \sqrt{}) [x := 1 \parallel (x := 0; x := x + 1)] (d[x=1], x := 0; x := x + 1) \\ &\quad \vdots \\ &(d, \sqrt{}) [x := 1 \parallel (x := 0; x := x + 1)] (d[x=2], \sqrt{}). \end{aligned}$$

Very roughly (i.e., up to one-step transitions), (5) is related to (4) as follows

$$\begin{aligned} &(d_1, p_1) [p] (d_2, p_2) \text{ iff } p_1 \parallel p \xrightarrow{4}^{(d_1, d_2)} p_2 \\ &\quad p \xrightarrow{4}^{(d, d')} p' \text{ iff } (d, \sqrt{}) [p] (d', p'). \end{aligned}$$

While the idea that transitions can compose to produce transitions is a natural one, work on transition systems has tended to focus on atomic labels. A notable exception is Boudol and Castellani [5], where “at each step, the performed action is

a compound one, namely a labelled poset, not just an atom" (p. 25). The work below is most definitely related to Boudol and Castellani [5], and, going back further, to Plotkin [16], where *resumptions* from a domain Y satisfying

$$Y = D_{\perp} \rightarrow Pow(D_{\perp} + (D_{\perp} \times Y))$$

are considered. A simple connection between transition systems and P_1 can be established by interpreting P_1 in a domain X satisfying the equation

$$X = Pow((D \times (X + \{\sqrt{\cdot}\})) \times (D \times (X + \{\sqrt{\cdot}\}))) . \quad (6)$$

A rough way (ignoring \perp) of relating (6) to Y is by a map from $y \in Y$ to $x_y \in X$ such that

$$y(d) \approx \{d' \mid (d, \sqrt{\cdot}) x_y (d', \sqrt{\cdot})\} + \{(d', y') \in D \times X \mid (d, \sqrt{\cdot}) x_y (d', x_{y'})\} ,$$

following the associations

$$\begin{aligned} D \rightarrow Pow(D + (D \times Y)) &\approx (D \times \{\sqrt{\cdot}\}) \rightarrow Pow((D \times \{\sqrt{\cdot}\}) + (D \times Y)) \\ &\approx (D \times \{\sqrt{\cdot}\}) \rightarrow Pow(D \times (Y + \{\sqrt{\cdot}\})) . \end{aligned}$$

Section 3 describes a "final" model for (6), without appealing to domain theory or introducing \perp . The essential complication addressed is the indefinite character of equality brought on by circularity — a difficulty resolved below by a quotient construction that, if not standard, is at least closely related to such (Milner [13]).

Remark. To confuse the symbol $\sqrt{\cdot}$ with some "no-op" $skip \in A$ (relating identical data states) is to invite trouble in grounding the circular notion of a program as a relation on $S = D \times (P_1 + \{\sqrt{\cdot}\})$. (See part 1 of Theorem 2 below.) The second component of such a state is, intuitively, a (finite) "bag" of programs to be executed — of which any number of instances of $skip$ may be included. The symbol $\sqrt{\cdot}$ is not a program, but is an indicator that the bag is empty. The identification of bags with elements of $P_1 + \{\sqrt{\cdot}\}$ is possible because the external request p in (5) is assumed to have as much right to be executed as any of the suspended internal programs (whence a non-empty bag can, through \parallel , be reduced to a single program). For a different scheduling policy (involving, for example, priorities), the rules given in the next section must be modified accordingly.

2 One transition system for five

To relate transitions (1) through (5), the transition system of Example 5 is extended below by taking $L = P_1 \times A^*$, with transitions of the form

$$(d_1, p_1) [p, \bar{a}] (d_2, p_2)$$

where $\bar{a} \in A^*$ records the sequence $a_1 \dots a_n$ of atomic programs actually executed. The idea is that (1) becomes

$$\exists \bar{a} (d, \sqrt{\cdot}) [p, \bar{a}] (d', \sqrt{\cdot}) ,$$

(2) becomes

$$\forall d \exists d' (d, \sqrt{} [p, a] (d', p') ,$$

(3) becomes

$$(d, \sqrt{} [p, a] (d', p') ,$$

(4) becomes

$$\exists a (d, \sqrt{} [p, a] (d', p') ,$$

and (5) becomes

$$\exists \bar{a} (d_1, p_1) [p, \bar{a}] (d_2, p_2) .$$

To be more precise, assume that among the a 's in A is *skip*, with $I_{skip} = \{(d, d) \mid d \in D\}$, and fix the following collection of rules, abusing notation so that $d, p, \theta, a, \bar{a}, \dots$ are understood as schematic variables ranging over $D, P_1, \Theta = P_1 \cup \{\sqrt{}\}, A, A^*, \dots$, respectively, and $a \in A$ is identified with the sequence of length one consisting of a . For every $a \in A$ and $(d, d') \in I_a$, throw in the axiom

$$(d, a, d') \quad \overline{(d, \sqrt{} [a, a] (d', \sqrt{})}$$

and close these transitions under

$$(\text{;}) \quad \frac{(d, \sqrt{} [p_1, \bar{a}] (d_1, \sqrt{}) \quad (d_1, \sqrt{} [p_2, \bar{b}] (d', \theta))}{(d, \sqrt{} [p_1; p_2, \bar{a}\bar{b}] (d', \theta))}$$

$$(+r) \quad \frac{(d, \sqrt{} [p_1, \bar{a}] (d', \theta))}{(d, \sqrt{} [p_1 + p_2, \bar{a}] (d', \theta))} \quad (l+) \quad \frac{(d, \sqrt{} [p_2, \bar{b}] (d', \theta))}{(d, \sqrt{} [p_1 + p_2, \bar{b}] (d', \theta))}$$

$$(*) \quad \frac{(d, \sqrt{} [skip + p; p^*, \bar{a}] (d', \theta))}{(d, \sqrt{} [p^*, \bar{a}] (d', \theta))}$$

$$(\|i) \quad \frac{(d, \sqrt{} [p, \bar{a}] (d', \sqrt{}))}{(d, p') [p, \bar{a}] (d', p')}$$

$$(\|-r) \quad \frac{(d, \sqrt{} [p, \bar{a}] (d', p_1))}{(d, p_2) [p, \bar{a}] (d', p_1 \| p_2)} \quad (l\|-) \quad \frac{(d, \sqrt{} [p, \bar{a}] (d', p_1))}{(d, p_2) [p, \bar{a}] (d', p_2 \| p_1)}$$

$$(\text{sym}) \quad \frac{(d, p_1) [p_2, \bar{a}] (d', \theta)}{(d, p_2) [p_1, \bar{a}] (d', \theta)}$$

$$(\text{shift}) \quad \frac{(d, p_1) [p_2, \bar{a}] (d', \theta)}{(d, \sqrt{} [p_1 \| p_2, \bar{a}] (d', \theta))}$$

$$\begin{aligned}
& (\text{trans}) \frac{(d, \theta) [p, \bar{a}] (d_1, p_1) \quad (d_1, \sqrt{}) [p_1, \bar{b}] (d', \theta')}{(d, \theta) [p, \bar{a}\bar{b}] (d', \theta')} \\
& (; i) \frac{(d, \sqrt{}) [p_1, \bar{a}] (d', p)}{(d, \sqrt{}) [p_1; p_2, \bar{a}] (d', p; p_2)} \quad (; i') \frac{(d, \sqrt{}) [p_1, \bar{a}] (d', \sqrt{})}{(d, \sqrt{}) [p_1; p_2, \bar{a}] (d', p_2)} .
\end{aligned}$$

The rules (d, a, d') , $(;)$, $(+r)$, $(l+)$ and $(*)$ correspond exactly to dynamic logic's semantic clauses for regular constructs. The remaining rules have been introduced to insure that \parallel is closed under interleaving. This can be made precise, although (suppressing the sequences of atomic programs from the labels for the sake of simplicity) the following derivation of $(d, p') [p] (d_2, p_1 \parallel p_2)$ from $(d, \sqrt{}) [p] (d_1, p_1)$ and $(d_1, \sqrt{}) [p'] (d_2, p_2)$ should be sufficient to make the point

$$\frac{\frac{(d, \sqrt{}) [p] (d_1, p_1)}{(d, p') [p] (d_1, p_1 \parallel p')} \quad \frac{\frac{(d_1, \sqrt{}) [p'] (d_2, p_2)}{(d_1, p_1) [p'] (d_2, p_1 \parallel p_2)}}{(d_1, \sqrt{}) [p_1 \parallel p'] (d_2, p_1 \parallel p_2)}}{(d, p') [p] (d_2, p_1 \parallel p_2)}$$

The reader interested in notions of parallelism connected with synchronization is referred to Appendix B.

For the record, the transition system defined above is $\langle P_1 \times A^*, D \times \Theta, \implies \rangle$ where \implies is

$$\{((d, \theta), (p, \bar{a}), (d', \theta')) \mid (d, \theta) [p; \bar{a}] (d', \theta') \text{ is derivable from the rule set above} \} .$$

Observe that whenever $(d_1, p_1) [p, a_1 \dots a_n] (d_2, p_2)$ is derivable, then $d_1 I_{a_1} \circ \dots \circ I_{a_n} d_2$. Other than the axioms (d, a, d') , the only rules that change the sequences of atomic programs in a transition (or the data-states related) are the "transitive" rules $(;)$ and (trans) , both of which increase the length of sequences. Neither has a counterpart in Examples 2 through 4, which are confined to sequences in A^* of length one. Thus, the rules $(;)$ and (trans) can be introduced "conservatively" into Examples 2 through 4, since these are rendered irrelevant by the restriction to transitions with atomic program sequences of length one. (The addition of these rules takes on some significance, however, if rules reducing the length of atomic program sequences in transitions are subsequently introduced — see the discussion in the concluding section.) Thus, as pointed out informally at the beginning of this section, the sets of transitions for Examples 1, 3, 4 and 5 can be described by the following definitions, where an additional example lying between Examples 4 and 5 (and so denoted 4.5) is included for later reference

$$\begin{aligned}
\llbracket \cdot \rrbracket_0 &= \{(d, p, d') \mid (d, \sqrt{}) [p, \bar{a}] (d', \sqrt{}) \text{ is derivable from the axioms } (d_0, a_0, d'_0) \\
&\quad \text{by } (;), (+r), (l+), \text{ and } (*), \text{ for some } \bar{a}\} \\
\rightarrow_3 &= \{((p, d), a, (\theta', d')) \mid (d, \sqrt{}) \xrightarrow{p, a} (d', \theta')\} \\
\rightarrow_4 &= \{(p, (d, d'), \theta') \mid (d, \sqrt{}) \xrightarrow{p, a} (d', \theta') \text{ for some } a\} \\
\rightarrow_{4.5} &= \{(p, (d, d'), \theta') \mid (d, \sqrt{}) \xrightarrow{p, \bar{a}} (d', \theta') \text{ for some } \bar{a}\} \\
\rightarrow_5 &= \{((d, \theta), p, (d', \theta')) \mid (d, \theta) \xrightarrow{p, \bar{a}} (d', \theta') \text{ for some } \bar{a}\} .
\end{aligned}$$

We show next that \Rightarrow extends $\llbracket \cdot \rrbracket_0$ “conservatively” in the sense that for every $p \in P_0$ and all $d, d' \in D$, if $(d, \sqrt{}) \xrightarrow{p, \bar{a}} (d', \sqrt{})$ for some \bar{a} , then $d \llbracket p \rrbracket_0 d'$. For this purpose, it is convenient to work with rules where the sequences of atomic programs in labels are erased, so that $L = P_1$ and transitions have the form of (5). Call the resulting set of rules Γ_1 , and let Γ_0 be the subset of Γ_1 given by (d, a, d') ’s, $(;)$, $(+r)$, $(l+)$, and $(*)$ with the atomic sequences dropped from the labels. It is easy to see that

$$\begin{aligned} \llbracket \cdot \rrbracket_0 &= \{(d, p, d') \mid (d, \sqrt{}) [p] (d', \sqrt{}) \text{ is derivable from } \Gamma_0\} \\ \rightarrow_5 &= \{((d, \theta), p, (d', \theta')) \mid (d, \theta) [p] (d', \theta') \text{ is derivable from } \Gamma_1\}. \end{aligned}$$

Theorem 1. \Rightarrow is a conservative extension of dynamic logic: that is, for every program $p \in P_0$,

$$(d, \sqrt{}) [p] (d', \sqrt{}) \text{ follows from } \Gamma_0 \text{ iff } (d, \sqrt{}) [p] (d', \sqrt{}) \text{ follows from } \Gamma_1.$$

Proof. Only (\Leftarrow) requires an argument. First, establish by induction on the length of Γ_1 -derivations that

- (†) For every $p \in P_0$ and every Γ_1 -derivation of $(d, \sqrt{}) [p] (d', \theta)$, it is the case that $\theta \in P_0 \cup \{\sqrt{}\}$ and the rules (sym), (shift), $(\|i)$, $(\|r)$, and $(l-\|)$ do not occur in the derivation.

Next, to push through an induction on the length of Γ_1 -derivations, it is useful to strengthen the induction hypothesis as follows. Given a Γ_1 -derivation D of $(d, \sqrt{}) [p] (d', \theta)$, let $\varphi(D)$ be the assertion

$$\begin{aligned} &\text{if } \theta = \sqrt{}, \text{ then } \Gamma_0 \text{ proves that } (d, \sqrt{}) [p'] (d', \sqrt{}), \\ &\text{else if } \theta = p' \text{ and } \Gamma_0 \text{ proves that } (d', \sqrt{}) [p'] (d'', \sqrt{}), \text{ then } \Gamma_0 \text{ proves that} \\ &(d, \sqrt{}) [p] (d'', \sqrt{}). \end{aligned}$$

Now, induct on the length of D , noting from (†) that $\theta \in P_0 \cup \{\sqrt{}\}$, and that (sym), (shift), $(\|i)$, $(\|r)$, and $(l-\|)$ do not occur in such a derivation. The argument breaks up into different cases, according to the last rule applied in D .

Case 1: rule from Γ_0 . Apply the induction hypothesis to the derivation of the premise of the last step; then use the same rule to prove $\varphi(D)$.

Case 2: (trans). D consists of the application of (trans) to a Γ_1 -derivation D_0 of $(d, \sqrt{}) [p] (d_1, p_1)$ and a Γ_1 -derivation D_1 of $(d_1, \sqrt{}) [p_1] (d', \theta)$. To prove $\varphi(D)$, suppose $\theta = p'$ and Γ_0 proves that $(d', \sqrt{}) [p'] (d'', \sqrt{})$. Then by $\varphi(D_1)$, it follows that Γ_0 proves $(d_1, \sqrt{}) [p_1] (d'', \sqrt{})$. Conclude from $\varphi(D_0)$ that $(d, \sqrt{}) [p] (d'', \sqrt{})$ follows from Γ_0 , as required. The case where $\theta = \sqrt{}$ is even simpler.

Case 3: $(; i)$. Then p is of the form $p_1; p_2$, and $\theta = p'$ is of the form $p_0; p_2$, where the premise of the last step is $(d, \sqrt{}) [p_1] (d', p_0)$. Call the subderivation of that premise D' . To prove $\varphi(D)$, suppose Γ_0 proves that $(d', \sqrt{}) [p'] (d'', \sqrt{})$. Since p' is $p_0; p_2$ (and Γ_0 yields dynamic logic), it follows that for some d''' , Γ_0 proves $(d', \sqrt{}) [p_0] (d''', \sqrt{})$ and $(d''', \sqrt{}) [p_2] (d'', \sqrt{})$. By $\varphi(D')$, deduce that Γ_0 proves $(d, \sqrt{}) [p_1] (d''', \sqrt{})$, whence by $(;)$, Γ_0 proves $(d, \sqrt{}) [p_1; p_2] (d'', \sqrt{})$, as desired.

Case 4: $(; i')$. similar to Case 3. \dashv

Having established Theorem 1, notice that the rule $(;)$ is derivable from (trans) and $(; i')$.

3 Semantic foundations for ‘programs as relations’

This section works out a semantics for programs based on (P1), “a program as a relation on states”, over a transition system $\langle P_1, D \times \Theta, [\cdot]_R \rangle$ whose transition relation is given by some rule set Γ (such as Γ_0 or Γ_1) as follows

$$(d, \theta) [p]_R (d', \theta') \text{ iff } (d, \theta) [p] (d', \theta') \text{ follows from } \Gamma.$$

Notice that “ $(d, \theta) [p] (d', \theta')$ ” is viewed in the right hand side above as a syntactic expression — a practice that we will adapt for the remainder of the paper. Formally, the idea is to work in a logical system without equality, but with (atomic) predicate symbols

$$(d, \cdot) [\cdot] (d', \cdot), (d, \sqrt{\cdot}) [\cdot] (d', \cdot), (d, \cdot) [\cdot] (d', \sqrt{\cdot}), (d, \sqrt{\cdot}) [\cdot] (d', \sqrt{\cdot})$$

for every d and $d' \in D$. The holes \cdot are to be filled by programs in P_1 . Now, a “semantics” for the set P_1 of programs is at the very least a map f with domain P_1 . One might expect this map to be defined by induction on terms (as in first-order logic), but for the time being, we will simply take f to be given, and will return to the matter of compositionality later. For f to be, in any sense, a model of Γ , all transitions $(d, \theta_0) [p]_R (d', \theta'_0)$ must hold in f . Without saying anything about the co-domain of f , let us focus on the “equality” relation $\{(p, p') \in P_1 \times P_1 \mid f(p) = f(p')\}$ on P_1 that f induces. While our list of predicate symbols does not include equality, the principle of “substituting equals for equals” can be implemented by closing $[\cdot]_R$ under equality

$$(d, \theta) [p] (d', \theta') \text{ is } \Gamma\text{-true in } f \text{ iff } \exists \theta_0, \theta'_0 \ f(\theta_0) = f(\theta), \ f(\theta'_0) = f(\theta') \text{ and } (d, \theta_0) [p]_R (d', \theta'_0),$$

where, for notational convenience, f has been extended to Θ by setting $f(\sqrt{\cdot}) = \sqrt{\cdot}$. Now, a minimal¹ condition of soundness on f with respect to Γ is that if f identifies p with p' then for every $d, d' \in D$ and $\theta, \theta' \in \Theta$,

$$(d, \theta) [p] (d', \theta') \text{ is } \Gamma\text{-true in } f \text{ iff } (d, \theta) [p'] (d', \theta') \text{ is } \Gamma\text{-true in } f.$$

In other words, if $[f]_R$ is defined as the map from P_1 given by

$$[f]_R(p) = \{((d, \theta), (d', \theta')) \mid (d, \theta) [p] (d', \theta') \text{ is } \Gamma\text{-true in } f\},$$

then

$$(\star) \ \forall p, p' \in P_1 \ f(p) = f(p') \text{ implies } [f]_R(p) = [f]_R(p'),$$

or, equivalently,

¹ As pointed out to the author by P. Panangaden, coarser notions of equivalence may for various purposes be desirable. But not every equivalence can be considered an “equality” from the point of view of a logical system given by the (atomic) predicate symbols $(d, \cdot) [\cdot] (d', \cdot)$, $(d, \sqrt{\cdot}) [\cdot] (d', \cdot)$, $(d, \cdot) [\cdot] (d', \sqrt{\cdot})$, and $(d, \sqrt{\cdot}) [\cdot] (d', \sqrt{\cdot})$ for every d and $d' \in D$. Of course, one can, as in logic, consider restrictions (i.e., “reducts”) of this language, or translate from this language to another — for example, translating a program p to its atomization $[p]$ (discussed in section 5) would abstract out intermediate states, and identify programs with the same input-output behavior. While Theorem 2 below holds quite generally, the congruence and soundness results in the appendix do not.

(\star)' for all $p, p', d_1, d_2, \theta_1, \theta_2$ such that $f(p) = f(p')$ and $(d_1, \theta_1) [p]_r (d_2, \theta_2)$, there exist θ'_1 and θ'_2 such that $f(\theta'_1) = f(\theta_1)$, $f(\theta'_2) = f(\theta_2)$ and $(d_1, \theta'_1) [p']_r (d_2, \theta'_2)$.

A second condition to impose on f is

$$(\star\star) \quad \forall \theta \in \Theta \quad f(\theta) = \sqrt{\quad} \text{ iff } \theta = \sqrt{\quad}.$$

This condition was justified conceptually by the final remark in section 1.5 above, and can be motivated technically by part 1 of Theorem 2 below. In any case, call a function f with domain Θ satisfying (\star) and ($\star\star$) Γ -consistent. Assuming that $\sqrt{\quad} \notin P_1$ (as we do throughout the paper), the identity id on Θ is Γ -consistent. A more interesting example arises from attempting to satisfy (\star) trivially by asking for an f which, restricted to P_1 , is $[f]_r$.

There are two possible complications with this request. One is that such f 's might fail to exist — which is precisely the case for the usual universe of sets satisfying the axiom of foundation. So let's drop the axiom of foundation. The second complication is that there may be different f 's for which $f = [f]_r$. While this "complication" may not seem so terrible, it is this very point that makes the domain equation (6) problematic (as the axiom of extensionality is no longer sufficient to settle questions of identity once circularity is admitted). It turns out to be convenient to appeal to the *Anti-Foundation Axiom* (AFA) in Aczel [1] asserting the uniqueness of an f for which $f = [f]_r$. (Readers familiar with Aczel [1] can see this by noting that for every $p \in P_1$,

$$[f]_r(p) = \{((d, f(\theta)), (d', f(\theta'))) \mid (d, \theta) [p]_r (d', \theta')\},$$

and that $\sqrt{\quad}$ can certainly be assumed not to be a set of ordered pairs.) Working in a universe of sets satisfying AFA and the usual set-theoretic axioms minus foundation, let's call that unique solution $[\cdot]_r$. One way to see the importance of $[\cdot]_r$ is through a little category theory.

Form a category \mathbf{C}_Γ with Γ -consistent functions as objects as follows. For a Γ -consistent function f , a function α_Γ^f with domain $f''P_1$ (i.e., the image $\{f(p) \mid p \in P_1\}$ of P_1 under f) can be defined by requiring

$$\alpha_\Gamma^f(f(p)) = [f]_r(p)$$

since f satisfies (\star). The \mathbf{C}_Γ -morphisms from f to g are (by definition) the functions φ from $f''\Theta$ to $g''\Theta$ such that $\varphi(\sqrt{\quad}) = \sqrt{\quad}$ and for every $p \in P_1$,

$$\alpha_\Gamma^g(\varphi f p) = \{((d, \varphi f \theta), (d', \varphi f \theta')) \mid (d, \theta) [p]_r (d', \theta')\}.$$

This equation can be pictured as follows

$$\begin{array}{ccccc} P_1 & \xrightarrow{f} & f''P_1 & \xrightarrow{\alpha_\Gamma^f} & F(f''P_1) \\ & & \downarrow \varphi & & \downarrow F(\varphi) \\ P_1 & \xrightarrow{g} & g''P_1 & \xrightarrow{\alpha_\Gamma^g} & F(g''P_1) \end{array}$$

where F is the functor on classes X (and class maps) given by

$$F(X) = \text{Pow}((D \times (X + \{\sqrt{\cdot}\})) \times (D \times (X + \{\sqrt{\cdot}\})))$$

together with the obvious map on morphisms (Aczel [1]).

Next, let $\theta \sim_\Gamma \theta'$ abbreviate $\llbracket \theta \rrbracket_\Gamma = \llbracket \theta' \rrbracket_\Gamma$, and for every Γ -consistent f , define the function $[f]_\Gamma^0$ with domain P_1 to record the input-output behavior of a program

$$\begin{aligned} [f]_\Gamma^0(p) &= \{(d, d') \mid (d, \sqrt{\cdot}) [p] (d', \sqrt{\cdot}) \text{ is } \Gamma\text{-true in } f\} \\ &= \{(d, d') \mid (d, \sqrt{\cdot}) [f]_\Gamma(p) (d', \sqrt{\cdot})\} . \end{aligned}$$

Also, let us understand “the theory” of a Γ -consistent object f to mean the set of all expressions $(d, \theta)[p](d', \theta')$ Γ -true in f . The following theorem records some pleasant logical properties enjoyed by \mathbf{C}_Γ , the first of which expresses that the input-output behavior prescribed by Γ is respected.

Theorem 2.

1. For every \mathbf{C}_Γ -object f , $[f]_\Gamma^0(p) = [\text{id}]_\Gamma^0(p)$.
2. \mathbf{C}_Γ -morphisms preserve truth; that is, if there is a \mathbf{C}_Γ -morphism from f to g mapping $f(p)$ to $g(p')$ then $[f]_\Gamma(p) \subseteq [g]_\Gamma(p')$.
3. id is initial in \mathbf{C}_Γ , and has the least theory of all objects in \mathbf{C}_Γ .
- 4 (AFA). At the other extreme, $\llbracket \cdot \rrbracket_\Gamma$ is final in the category \mathbf{C}_Γ . It has the largest theory of all objects in \mathbf{C}_Γ .
- 5 (AFA). For $p, p' \in P_1$,

$$p \sim_\Gamma p' \Leftrightarrow \text{there is a } \Gamma\text{-consistent function } f \text{ with } f(p) = f(p') .$$

Proof. Unwrap the definitions, which, in the case of $\llbracket \cdot \rrbracket_\Gamma$, includes a uniqueness property. In particular, note that for every \mathbf{C}_Γ -object f , φ is a \mathbf{C}_Γ -morphism from f to $\llbracket \cdot \rrbracket_\Gamma$ iff $\varphi \circ f = \llbracket \cdot \rrbracket_\Gamma$. \dashv

So long as one is content with “a program *isomorphic* to a relation on states”, it is only the equivalence $\{(p, p') \mid f(p) = f(p')\}$ induced by an interpretation f that matters. Thus, having characterized \sim_Γ above with the help of AFA, it is possible to forget $\llbracket \cdot \rrbracket_\Gamma$ and AFA, and “simply” show that the function sending $\sqrt{\cdot}$ to $\sqrt{\cdot}$ and p to

$$\{p' \in P_1 \mid p \sim_\Gamma p'\}$$

is final in \mathbf{C}_Γ . It is the categorical property of finality that is interesting, not only because it captures a certain maximality formulated in part 4 of the theorem above, but also because it turns out to be useful in establishing that the equality induced is a congruence, provided Γ has a “nice” form. The technical details have been relegated to an appendix, since, as it happens, the case of Γ_1 is reducible to results from Groote and Vaandrager [11]. One other point that Appendix A attends to is that Γ is indeed sound for $\llbracket \cdot \rrbracket_\Gamma$ (under certain assumptions on Γ).

4 Relating the examples semantically

Underlying Examples 1 and 2, on which the other examples build, are the conceptions (P1) and (P2). Preserving (P1), Example 5 extends Example 1 (conservatively) by attaching programs to data-states. Similarly, Example 3 attaches data-states to Example 2's process-states, while Example 4 replaces atomic program labels with data-state pairs (from the atomic programs). The question arises as to whether (P2) and (P1) are fundamentally different. To attack this question semantically, we consider "natural" models for the examples above, focussing on the relations of "equality" induced on the programs in P_1 .

The case of Example 5 has already been treated in the previous section; summarizing (and suppressing Γ_1 from the notation for simplicity), define

$$p \sim p' \text{ iff there is a } \Gamma_1\text{-consistent } f \text{ such that } f(p) = f(p')$$

where f is Γ_1 -consistent iff it is a function with domain Θ meeting conditions

(\star)' for all $p, p', d_1, d_2, \theta_1, \theta_2$ such that $f(p) = f(p')$ and $(d_1, \theta_1) \xrightarrow{\beta_5} (d_2, \theta_2)$, there exist θ'_1 and θ'_2 such that $f(\theta'_1) = f(\theta_1)$, $f(\theta'_2) = f(\theta_2)$ and $(d_1, \theta'_1) \xrightarrow{\beta'_5} (d_2, \theta'_2)$,

and

($\star\star$) for all $\theta \in \Theta$, $f(\theta) = \surd$ iff $\theta = \surd$.

As for (P2), a standard notion of equivalence due to Park [14] is given as follows (modified slightly to take \surd into account). Define a *bisimulation* on a transitions system $\langle L, S, \rightarrow \rangle$ to be a relation $R \subseteq S \times S$ such that whenever sRs' , then for every $l \in L$,

$$\forall t \leftarrow s \exists t' \leftarrow s' tRt' \text{ and } \forall t' \leftarrow s' \exists t \leftarrow s tRt',$$

and $s = \surd$ iff $s' = \surd$. The $\forall\exists$ -conjunctions above should be compared with condition (\star)' for Γ -consistent functions, noting that the symmetry of the predicate $f(p) = f(p')$ renders the other half of the back-and-forth clause for (\star)' unnecessary. Next, define the *bisimilarity* relation \leftrightarrow on $\langle L, S, \rightarrow \rangle$ by

$$s \leftrightarrow s' \text{ iff there is a bisimulation on } \langle L, S, \rightarrow \rangle \text{ relating } s \text{ to } s'.$$

The semantics studied in BKPR [6] does not build on bisimulations, but, as pointed out to the author by J. Rutten, it follows from Groote and Vaandrager [11] (see also Rutten [17]) that the bisimilarity predicate \leftrightarrow_4 for Example 4

$$\theta \leftrightarrow_4 \theta' \text{ iff there is a bisimulation on } \langle D \times D, \Theta, \rightarrow_4 \rangle \text{ relating } \theta \text{ to } \theta'$$

is a congruence. This property is preserved by the addition of the rule (trans); that is, the predicate $\leftrightarrow_{4.5}$ for $\langle D \times D, \Theta, \rightarrow_{4.5} \rangle$ is a congruence.

Theorem 3. $\leftrightarrow_{4.5}$ is equal to \sim .

Proof. That $p \sim p'$ implies $p \leftrightarrow_{4.5} p'$ follows from the fact that \sim is a bisimulation for $\langle D \times D, \Theta, \rightarrow_{4.5} \rangle$. Conversely, define a function f with domain Θ by setting

$f(\sqrt{}) = \sqrt{}$ and for $p \in P_1$, $f(p) = \{\theta \mid p \leftrightarrow_{4.5} \theta\}$. It is easy to see that if $p \leftrightarrow_{4.5} p'$ then f witnesses $p \sim p'$ because, as noted above, $\leftrightarrow_{4.5}$ is a congruence with respect to \parallel . \dashv

Adding the rule (trans) to Example 4 represents a real change in that, for instance, $a + \text{skip}; a \sim \text{skip}; a$ (although $\text{skip}; \text{skip} \not\sim \text{skip}$) while Example 4 differentiates between these when $I_a \neq I_{\text{skip}}$. Whereas the step from Example 1 to Example 4 shows that bisimilarity can become finer with an increase in rules, the move from \rightarrow_4 to $\rightarrow_{4.5}$ shows that it can also become coarser when the rule set is extended. (Note that the transition predicate occurs both positively and negatively in the definition of a bisimulation.)

Theorem 4. \leftrightarrow_4 is (in general, strictly) finer than \sim .

Proof. Following the previous proof, define a function f with domain Θ by setting $f(\sqrt{}) = \sqrt{}$ and for $p \in P_1$, $f(p) = \{\theta \mid p \leftrightarrow_4 \theta\}$. Next, to see that f is Γ_1 -consistent, it suffices to show that

(†) if a transition $(d, \theta) [p] (d', \theta')$ can be derived in Γ_1 , then there are $d_1, p_1, d_2, p_2, \dots, d_k, p_k$ such that $(d, \theta) \xrightarrow{p}_4 (d_1, p_1)$, $(d_1, \sqrt{}) \xrightarrow{p_1}_4 (d_2, p_2), \dots, (d_k, \sqrt{}) \xrightarrow{p_k}_4 (d', \theta')$, whence repeated applications of (trans) yield $(d, \theta) [p]_{\Gamma_1} (d', \theta')$.

The assertion (†) holds because (i) the rule (;) can be replaced by (trans) and (;'), and (ii) the rule (trans) can always be moved below a rule applied after it — for example, by locally converting the result of (trans) and (*) applied in that order

$$\frac{(d, \theta) [\text{skip} + p; p^*] (d_1, p') \quad (d_1, \sqrt{}) [p'] (d', \theta')}{\frac{(d, \theta) [\text{skip} + p; p^*] (d', \theta')}{(d, \theta) [p^*] (d', \theta')}} \quad (d, \theta) [p] (d', \theta')$$

to the result of (*) and then (trans)

$$\frac{\frac{(d, \theta) [\text{skip} + p; p^*] (d_1, p')}{(d, \theta) [p^*] (d_1, p')} \quad (d_1, \sqrt{}) [p'] (d', \theta')}{(d, \theta) [p] (d', \theta')}$$

Note: a rule called (at) is introduced in the next section with which (trans) cannot commute as above. \dashv

Similarly, Example 3 is even less abstract than Example 4. A *global D-bisimulation* on $\langle A, P \times D, \rightarrow_3 \rangle$ is defined in Groote and Ponse [10] to be a relation $R \subseteq (P \times D) \times (P \times D)$ such that whenever $(p, d)R(p', d)$, then for every $a \in A$,

$$\begin{aligned} \forall (p_1, d_1) \xleftarrow{a}_3 (p, d) \exists (p'_1, d_1) \xleftarrow{a}_3 (p', d) \forall d_2 (p_1, d_2)R(p'_1, d_2) \quad \text{and} \\ \forall (p'_1, d_1) \xleftarrow{a}_3 (p', d) \exists (p_1, d_1) \xleftarrow{a}_3 (p, d) \forall d_2 (p_1, d_2)R(p'_1, d_2). \end{aligned}$$

The difference in the treatment of data from that of a process might be defended by appealing to the intuition that data is “irreducible” in a sense that a process is not. As it turns out, however, even for (1), the notion of a bisimulation is interesting, a point to which we will return in the final section. In any case, let

$p \leftrightarrow_3 p'$ iff for all $d \in D$ there is a global D -bisimulation R such that $(p, d)R(p', d)$.

Theorem 5. \leftrightarrow_3 is (in general, strictly) finer than \leftrightarrow_4 .

Proof. If $p \xrightarrow{(d,d')}_4 p'$, then $(p, d) \xrightarrow{a}_3 (p', d')$ for some a with $dI_a d'$. So if R is a global D -bisimulation relating (p, d) to (p_0, d) then for some (p'_0, d') $\xrightarrow{a}_3 (p_0, d)$, R relates (p', d') to (p'_0, d') for every d' . But in that case, $p_0 \xrightarrow{(d,d')}_4 p'_0$. Hence, $p \leftrightarrow_3 p'$ implies $p \leftrightarrow_4 p'$.

As for the reverse direction, consider $a_1 + a_2$ and a when $I_{a_1} \cup I_{a_2} = I_a$ (or, to see that strictness can result even when every I_a is a distinct total function, $a_3 + a_4$ where $I_{a_1} \cup I_{a_2} = I_{a_3} \cup I_{a_4}$). \dashv

5 Discussion

As is made precise by Theorems 3 and 4, the key element setting Example 5 (the synthesis of (P1) and (P2)) apart from Examples 2 through 4 is the explicit incorporation into transitions of the composition rule (trans). If the rule (trans) increases the length of an atomic sequence labelling a transition, then what about a dual rule decreasing the length of the sequence? An example is provided by adding a closure clause for the “atomization” $[p]$ of p to P_1 , together with the rule

$$(\text{at}) \frac{(d, \sqrt{}) [p, \bar{a}] (d', \sqrt{})}{(d, \sqrt{}) [[p], [p]] (d', \sqrt{})} ,$$

resetting L to $P \times (A \cup \{[p] \mid p \in P\})^*$. (De Bakker and De Vink [3] provide a different treatment; the rule above is similar to one proposed independently by Franck van Breugel.) The congruence and soundness results described in Appendix A easily adapt to cover this extension.

Intuitively, the rule (trans) functions as the computational “glue” between transitions, possessing something of the transitive character of the *cut* rule in logic. Accordingly, a proof that a particular extension of the rule set of section 2 is, in some sense, conservative (e.g., Theorem 1) will likely involve an induction principle in which the rule (trans) plays a crucial role. The (i) inevitability of such extensions, and the (ii) measure of computational content (either of the initial system or of the extension) the induction principles provide are topics that the present author plans to address in future work. The line of thinking here is motivated largely by the prevailing view behind applications of proof theory to programming language semantics that computation is related in a deep sense to cut-elimination (or more broadly, to deduction). This view would be a bit more convincing if it can be shown to shed light on computations that need not terminate.

To analyze the sort of computational processes considered above, it might well be useful to pass from natural numbers to sets — i.e., to adapt ordinary recursion theory to sets. Transition systems are afterall sets, or at least can be coded as such, much more naturally than as natural numbers, given the extensional character of the notions of identity involved. Furthermore, the usual set-theoretic coding of, for example, ordered pairs can be employed even in the presence of circularity, by appealing to the theory of non-well-founded sets of Aczel [1]. The notion of equality

underlying AFA is, in fact, bisimilarity (simplified to the case where L is a singleton). In this respect, the “independent” derivation of the notion of bisimulation described in section 3 is perhaps not so surprising. The essential point, in any case, is that set theories encode the ontological commitments of a conception in a particularly parsimonious manner (the theories involved requiring only a single binary relation symbol \in), making them ideal for foundational investigations — including computationally sensitive ones. In particular, the generalization of recursion theory to sets described in Barwise [4] has proved fruitful for infinitary extensions of first-order logic, which might be employed for getting a logical grip on \sim (Fernando [7] [9]).

First-order logic is relevant also to the transitions (1) of Example 1, as natural examples of data-states and atomic programs are based on first-order models. Fernando [8] takes up just such cases, examining bisimulations on transition systems constructed from first-order models (as prescribed by dynamic logic). Whereas bisimulations are employed in process theory to analyze control, with data abstracted away, bisimulations are used in Fernando [8] to study variations in notions of data. (The latter is motivated by the absence of a natural choice of a first-order model for natural language semantics comparable, for example, to the first-order structure of arithmetic for programming language semantics.)

References

1. Peter Aczel. *Non-Well-Founded Sets*. CSLI Lecture Notes Number 14, Stanford, 1988.
2. J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.
3. J.W. de Bakker and E.P. de Vink. Bisimulation semantics for concurrency with atomicity and action refinement. Technical Report CS-R9210, Centre for Mathematics and Computer Science, 1992.
4. Jon Barwise. *Admissible Sets and Structures*. Springer-Verlag, Berlin, 1975.
5. G. Boudol and I. Castellani. Concurrency and atomicity. *Theoretical Computer Science*, 59, 1988.
6. 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 *Proc. Concur '91*, LNCS 527. Springer-Verlag, Berlin, 1991.
7. Tim Fernando. A primitive recursive set theory and AFA: on the logical complexity of the largest bisimulation. To appear in the proceedings of Computer Science Logic '91 (Berne).
8. Tim Fernando. Transition systems and dynamic semantics. To appear in the proceedings of JELIA '92 (Berlin).
9. Tim Fernando. Between programs and processes: absoluteness and open ended-ness. Technical Report IAM 92-011, Institut für Informatik, Universität Bern, 1992.
10. J.F. Groote and A. Ponse. Process algebra with guards: combining Hoare logic with process algebra. In *Proc. Concur '91*, LNCS 527. Springer-Verlag, Berlin, 1991.
11. J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. In *Proc. 16th ICALP*, LNCS 372. Springer-Verlag, Berlin, 1989.
12. David Harel. Dynamic logic. In Gabbay et al, editor, *Handbook of Philosophical Logic, Volume 2*. D. Reidel, 1984.
13. Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25, 1983.

14. David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI Conference*, LNCS 104. Springer-Verlag, Berlin, 1981.
15. David Peleg. Concurrent dynamic logic. *J. Assoc. Computing Machinery*, 34(2), 1987.
16. Gordon D. Plotkin. A powerdomain construction. *SIAM J. Comput.*, 5(3), 1976.
17. J.J.M.M. Rutten. Processes as terms: Non-well-founded models for bisimulation, 1992. To appear in *Mathematical Structures in Computer Science*.

Appendix A: a final congruence sound for ‘programs as relations’

This appendix addresses the matter of compositionality and soundness of Γ -consistent functions. Compositionality is trivial for the initial \mathbf{C}_Γ -object id , but is most interesting for the final \mathbf{C}_Γ -objects. The problem of interpreting the program constructs $;$, $+$, $*$ and \parallel over a Γ -consistent function f reduces to showing that the equality $\{(p, p') \in P_1 \times P_1 \mid f(p) = f(p')\}$ it induces is a congruence with respect to the constructs. The point is, for example, that $;$ can be interpreted under f in at most one way — viz.,

$$f(p) \bullet f(p') = f(p; p') .$$

This equation can be taken as a sound definition of \bullet iff it can be shown to be independent of the choice of representatives p and p' (i.e., iff $\{(p, p') \mid f(p) = f(p')\}$ is a congruence with respect to $;$). As is made clear in Groote and Vaandrager [11], this property becomes problematic for a semantic analysis based on syntactic rules, such as $(*)$ in section 2, without a “subformula property.”

The constraints imposed by Γ apply more directly to $[f]_\Gamma(p)$ than to $f(p)$, suggesting that it may be useful to restrict attention to \mathbf{C}_Γ -objects f for which α^f is 1-1 (i.e., for all p and $p' \in P_1$, $[f]_\Gamma(p) = [f]_\Gamma(p')$ implies $f(p) = f(p')$). But even this property is not strong enough to overcome obstacles posed by rules such as $(; i)$ to an argument (by induction on rules) that

$$f(p_1) = f(p'_1) \text{ and } f(p_2) = f(p'_2) \text{ imply } [f]_\Gamma(p_1; p_2) = [f]_\Gamma(p'_1; p'_2) .$$

The “strong extensionality” of $\llbracket \cdot \rrbracket_\Gamma$ is, however, sufficient for a wide collection of rules, as will be shown shortly.

Given a set X of program variables, let $P_1(X)$ be the collection of program terms constructed by applying $+$, $;$, $*$ or \parallel to (constants denoting) elements of A or program variables in X . (Thus, $P_1 = P_1(\emptyset)$.) Define R to be the following relation on P_1

$$\begin{aligned} \{ (t[p_1, \dots, p_n], t[p'_1, \dots, p'_n]) \mid n < \omega, t(x_1, \dots, x_n) \in P_1(\{x_1, \dots, x_n\}) \text{ and} \\ p_1 \sim_\Gamma p'_1, \dots, p_n \sim_\Gamma p'_n \} . \end{aligned}$$

Call a rule set Γ *nice* if for all $(p, p') \in R$, $\theta_1, \theta_2 \in \Theta$, and $d_1, d_2 \in D$ such that $(d_1, \theta_1) \llbracket p \rrbracket_\Gamma (d_2, \theta_2)$,

$$\begin{aligned} \exists \theta'_1, \theta'_2 \text{ s.t. } (\theta_1, \theta'_1) \in R \cup \{(\sqrt{}, \sqrt{})\} , (\theta_2, \theta'_2) \in R \cup \{(\sqrt{}, \sqrt{})\} \\ \text{and } (d_1, \theta'_1) \llbracket p' \rrbracket_\Gamma (d_2, \theta'_2) . \end{aligned}$$

Lemma A (AFA). For every nice rule set Γ , if $p_1 \sim_\Gamma p'_1$ and $p_2 \sim_\Gamma p'_2$, then $p_1^* \sim_\Gamma p'^*_1$ and $p_1 \circ p_2 \sim_\Gamma p'_1 \circ p'_2$ for $\circ \in \{;, +, \|\}$.

Proof. Since Γ is nice, it follows that the map f sending $\sqrt{}$ to $\sqrt{}$ and every $p \in P_1$ to $\{p' \mid pRp'\}$ is a \mathbf{C}_Γ -object. But by Theorem 2, $\llbracket \cdot \rrbracket_\Gamma$ is final, whence R is \sim_Γ , as desired. \dashv

Proving that the particular rule set Γ_1 is nice by induction on Γ_1 -derivations is complicated by the rules (sym) and (shift). (Try it.) For this reason, it is convenient to consider the stronger property

- (N) for all $(p, p') \in R$, $\theta_1, \theta_2 \in \Theta$, $d_1, d_2 \in D$, and θ'_1 s.t. $(\theta_1, \theta'_1) \in R \cup \{(\sqrt{}, \sqrt{})\}$,
if $(d_1, \theta_1) [p]_\Gamma (d_2, \theta_2)$ then $\exists \theta'_2$ s.t. $(\theta_2, \theta'_2) \in R \cup \{(\sqrt{}, \sqrt{})\}$ and
 $(d_1, \theta'_1) [p']_\Gamma (d_2, \theta'_2)$.

Property (N) can be proved by induction on the length of Γ -derivations assuming the rules in Γ have a certain form. To describe such a form, fix a countable set X of program variables, and call a program term in $P_1(X)$ *primitive* if it is either (a constant denoting) an element of A or a program variable ($\in X$). For $\hat{\theta} \in P_1(X) \cup \{\sqrt{}\}$, let $\text{Var}(\hat{\theta})$ be the set of variables in X occurring in $\hat{\theta}$. Now, consider a rule of the form

$$(r) \frac{(d_i, \theta_i) [p_i] (d'_i, \theta'_i) \quad (i < n)}{(d, \theta) [p] (d', \theta')}$$

where

- (c0) d_i, d'_i ($i < n$), d , and d' are (constants denoting) elements of D or variables ranging over D ,
 θ_i ($i < n$), and $\theta' \in P_1(X) \cup \{\sqrt{}\}$,
for $i < n$, $p_i \in P_1(X)$,
- (c1) θ and every θ'_i ($i < n$) are primitive program terms or $\sqrt{}$,
- (c2) $\text{Var}(\theta) \cap \text{Var}(p) = \emptyset$,
- (c3) $\text{Var}(\theta'_i) \cap \text{Var}(\theta'_j) = \emptyset$ for $i < j < n$,
- (c4) $\text{Var}(p_i) \cap \text{Var}(\theta'_j) = \text{Var}(\theta_i) \cap \text{Var}(\theta'_j) = \emptyset$ for $i \leq j < n$,
- (c5) $\text{Var}(p) \cap \text{Var}(\theta'_i) = \text{Var}(\theta) \cap \text{Var}(\theta'_i) = \emptyset$ for $i < n$, and
- (c6) p is either a primitive program term or x_1^* or $x_1 \circ x_2$ for some program variables x_1 and x_2 , and $\circ \in \{;, +, \|\}$.

Condition (c0) is not much of a restriction; the other conditions are best appreciated in the course of establishing congruence and soundness. Note that Γ_1 can be presented as a set of rules with the form of (r) — a rule with variables ranging over Θ can be broken up (by cases) into multiple rules of the form (r), using the fact that $\Theta = P_1 \cup \{\sqrt{}\}$.

Lemma B. If all rules in Γ have the form of (r) above, then property (N) holds and Γ is therefore nice.

Proof. Assume all rules in Γ have the form of (r) above. To prove that (N) holds, first establish

(†) whenever $(d_1, \theta_1) [p]_F (d_2, \theta_2)$ and $(\theta_1, \theta'_1) \in R \cup \{(\sqrt{\cdot}, \sqrt{\cdot})\}$, then there is a θ'_2 such that $(\theta_2, \theta'_2) \in R \cup \{(\sqrt{\cdot}, \sqrt{\cdot})\}$ and $(d_1, \theta'_1) [p]_F (d_2, \theta'_2)$.

A proof of (†) proceeds by induction on the length of F -derivations. Consider the last rule (r) of a shortest F -derivation of $(d_1, \theta_1) [p]_F (d_2, \theta_2)$. The point is that (c1) and (c2) allow the the premise to be re-instantiated appropriately with θ_1 replaced by θ'_1 , using the induction hypothesis and conditions (c3) to (c5). Then a θ'_2 can be chosen such that $(\theta_2, \theta'_2) \in R \cup \{(\sqrt{\cdot}, \sqrt{\cdot})\}$ and the conclusion of (r) can be instantiated by $(d_1, \theta'_1) [p]_F (d_2, \theta'_2)$.

Now, suppose $(p, p') \in R$, $\theta_1, \theta_2 \in \Theta$, $d_1, d_2 \in D$, $(\theta_1, \theta'_1) \in R \cup \{(\sqrt{\cdot}, \sqrt{\cdot})\}$, and $(d_1, \theta_1) [p]_F (d_2, \theta_2)$. If $p \sim_F p'$ then appeal to (†) above. Otherwise, adapt the induction argument for (†) using (c6) to replace p by p' . \dashv

Lemmas A and B yield

Theorem C (AFA). *If F is a set of rules with the form of (r) , then \sim_F is a congruence with respect to $;$, $+$, $*$ and \parallel .*

Note that the theorem applies to F_1 . The proof of Theorem C can easily be modified to cover expansions of P_1 , independent, as it is, of special properties of $;$, $+$, \parallel and $*$. On the other hand, the connection with the general congruence results of Groote and Vaandrager [11] is made above for the particular case of $;$, $+$, \parallel and $*$, with parallelism playing a central role in reducing Example 5 to Example 4 (via 4.5).

Finally, observe that another consequence of Lemma B (appealing to (N), (c3) and (c4)) is

Theorem D (AFA). *If F is a set of rules with the form of (r) , then F is sound for $[\cdot]_F$.*

Appendix B: living with ‘programs as relations’

The “parallel” construct \parallel formulated above captures interleaving. What about (as F. Vaandrager has asked) notions of synchronization say, on a set $H \subseteq A$ of atomic programs? Consider the binary program construct \parallel_H , described in the usual transition system format of Example 2 (without data) by the rules

$$(\alpha) \frac{p_1 \xrightarrow{a} p'_1 \quad p_2 \xrightarrow{a} p'_2}{p_1 \parallel_H p_2 \xrightarrow{a} p'_1 \parallel_H p'_2} \quad a \in H$$

$$(\beta) \frac{p_1 \xrightarrow{a} p'_1}{p_1 \parallel_H p_2 \xrightarrow{a} p'_1 \parallel_H p_2} \quad a \notin H.$$

Rules (α) and (β) do not specify how data-states are transformed. Presumably, (β) translates, in the format of section 2, to

$$\frac{(d, \sqrt{\cdot}) [p_1, a] (d', p'_1)}{(d, \sqrt{\cdot}) [p_1 \parallel_H p_2, a] (d', p'_1 \parallel_H p_2)} \quad a \notin H$$

but how about (α) ? One possibility is that there are functions i and j from D^4 to D , allowing (α) to be reformulated as

$$\frac{(d_1, \sqrt{}) [p_1, a] (d'_1, p'_1) \quad (d_2, \sqrt{}) [p_2, a] (d'_2, p'_2)}{(i(d_1, d'_1, d_2, d'_2), \sqrt{}) [p_1 \parallel_H p_2, a] (j(d_1, d'_1, d_2, d'_2), p'_1 \parallel_H p'_2)} \quad a \in H .$$

These rules present a problem for **(P1)** in that they are dependent on the specific atomic programs executed.

One way around this problem is to assume a sufficiently rich notion of data-state so as to be able to synchronize on data. It is true enough that data-states in dynamic logic only give values of program variables. But in the abstract set-up above, one can expand the notion to include “synchronization information” which the regular programs and interleaving \parallel will ignore. Pushing the operating system intuition mentioned in section 1.5 further, the idea is that some processes will operate on only a section of the computer’s memory. That is because other parts of the computer’s memory are devoted to matters of control.

More concretely, suppose D is the set of finite functions from some set $Store$ to some set of values, and that for every $a \in A$, we have an $I_a \subseteq D \times D$. Now, form a new set \hat{D} of data-states by adjoining a slot for A marked by some $\hat{s} \notin Store$ as follows

$$\hat{D} = \{d \cup \{(\hat{s}, a)\} \mid d \in D, a \in A\} .$$

Then for every $a \in A$, pass from I_a to

$$\hat{I}_a = \{(d \cup \{(\hat{s}, a')\}, d' \cup \{(\hat{s}, a)\}) \mid a' \in A, d I_a d'\} ,$$

so that the slot \hat{s} records the atomic program executed. Now, given an $H \subseteq A$, the rule (α) can be formulated as

$$(\alpha)' \quad \frac{(d_1, \sqrt{}) [p_1] (d'_1, p'_1) \quad (d_2, \sqrt{}) [p_2] (d'_2, p'_2)}{(d, \sqrt{}) [p_1 \parallel_H p_2] (d', p'_1 \parallel_H p'_2)} \quad S_H(d_1, d'_1, d_2, d'_2, d, d')$$

where

$$S_H(d_1, d'_1, d_2, d'_2, d, d') \text{ iff } d'_1(\hat{s}) = d'_2(\hat{s}) \in H, \quad d = d_1 \cup d_2, \quad d' = d'_1 \cup d'_2 .$$

Similarly, for (β) , take

$$(\beta)' \quad \frac{(d, \sqrt{}) [p_1] (d', p'_1)}{(d, \sqrt{}) [p_1 \parallel_H p_2] (d', p'_1 \parallel_H p'_2)} \quad d'(\hat{s}) \notin H .$$

The rule $(\alpha)'$ can be adapted for a synchronization construct $\&$ that is “continued” by a possibly different construct $[\cdot, \cdot]_{\&}$, but which is assumed to come with some predicate $S_{\&}$

$$(\alpha)'' \quad \frac{(d_1, \sqrt{}) [p_1] (d'_1, p'_1) \quad (d_2, \sqrt{}) [p_2] (d'_2, p'_2)}{(d, \sqrt{}) [p_1 \& p_2] (d', [p'_1, p'_2]_{\&})} \quad S_{\&}(d_1, d'_1, d_2, d'_2, d, d') .$$

(In Groote and Ponse [10], for example, note that the construct $|$ is continued by \parallel .) The possibilities are legion, but the essential point for the present paper is that

rules of the form $(\alpha)'$, $(\beta)'$ and $(\alpha)''$ are within the scope of the theory described in section 3 and Appendix A. (Side conditions such as $S_{\&}(d_1, d'_1, d_2, d'_2, d, d')$ can be eliminated by splitting the rule into the-cardinality-of- $S_{\&}$ -many rules.)

While the “trick” above is quite general, certainly not every notion of control we dream up can be accomodated *directly* in the formulation of **(P1)** by the domain equation (6). (And notice that *skip* is no longer interpreted as the identity on data-states by \hat{I}_{skip} , although the extension from D to \hat{D} and I_a to \hat{I}_a is, in a suitable sense, conservative.) The question is whether **(P1)** is conceptually natural enough that we should welcome the discipline it imposes on our thinking. A trade-off between **(P1)** and **(P2)** (especially as realized by Example 2) is mediated by data-states and atomic programs, with the passage $D, I_a \mapsto \hat{D}, \hat{I}_a$, suggesting a broad interpretation of “data” that includes atomic programs. But uniformity for uniformity’s sake can obscure simple ideas, and to insist (beyond observing that it can be done in principle) that a transition rule be put in a form with $L = P$ and $S = D \times (P + \{\sqrt{\}\})$ would be silly.