Comparative transition system semantics

Tim Fernando*
fernando@cwi.nl

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

Abstract. Employing the notion of a transition system, programs, conceived as binary (transition) relations on states, are related to processes, viewed as dynamic states. The comparative study is carried out syntactically over 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.

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 [11]) 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 $\langle \mathbf{R}, X, \rightsquigarrow \rangle$ 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 relations cannot (under the usual encoding of relations as sets). First, $\rightarrow$ allows two distinct labels l and l' to relate the same states

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

* 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. The 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').

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 (inasmuch as that account requires an analysis of intermediate states in a possibly non-terminating computation). 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 [13]).

Conceptually, then, 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. Atomization and synchronization are also 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 (more specifically, transition systems from Groote and Ponse [9], and De Boer, Kok, Palamidessi, and Rutten [6], which have been modified to highlight their essential characteristics — at least from the point of view of the present paper). 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" $d, d', d_1, \ldots$; P_1 is the set of "program(term)s" $p, \ldots$ given by

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

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 $\|$ is a bit more troublesome. A view of $\|$ is adopted throughout this paper, allowing the program $x := 1 \| (x := 0; x := x + 1)$ 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 $\|$. 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 $\|$. The semantic framework underlying dynamic logic provides a transition system where $L = P_0$ and $S = D$, with transitions written

$$d \, [\![p]\!]_0 \, d' \tag{1}$$

indicating that program p can yield (as output) d', given input d (and where $[\![a]\!]_0 = I_a$). (Note: the present paper concerns only the semantic component of dynamic logic, and not the modal logic for analyzing these structures.)

A well-known limitation of (1) is that it cannot support a compositional account of parallelism in that $[\![x := 1]\!]_0 = [\![x := 0; x := x + 1]\!]_0$, whereas $[\![x := 1 \| x := 1]\!]_0$ better not equal $[\![x := 1 \| (x := 0; x := x + 1)]\!]_0$. (The concurrent extension of dynamic logic in Peleg [14] where programs are interpreted as subsets of $D \times 2^D$ is inconsistent with this view.) Beyond the matter of compositionality, a direct analysis of intermediate states is suggested also by interest in non-terminating computations.

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 $L = A + \{\surd\}$ and $S = P_1 + \{\surd\}$, with transitions

$$p \xrightarrow{a} p' \tag{2}$$

indicating that process p can execute the atomic program a, and in so doing transform itself into p'. The symbol $\surd$ 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 [9], $L = A + \{\surd\}$, and $S = (P_1 + \{\surd\}) \times D$ (so-called "configurations"), with transitions of the form

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

extending Example 2 with the condition that $dI_a d'$. 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" $a \in A$, it is the case that $\forall d\ \exists d'\ \ d\ I_a\ d'$. Atomic programs that need not have this property are central to Groote and Ponse [9], 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 $S = P_1 + \{\surd\}$, with transitions

$$p \xrightarrow{d,d'}_4 p' \tag{4}$$

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 $p \xrightarrow{d_1,d'_1}_4 p_1 \xrightarrow{d_2,d'_2}_4 p_2 \cdots$ 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+i} 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 $L = P_1$, $S = D \times (P_1 + \{\surd\})$, and consider transitions

$$(d_1, p_1)\ [p\rangle\ (d_2, p_2)\ , \tag{5}$$

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 $p_1 = p_2 = \surd$, and $p \in P_0$.

External requests are assumed to have priority equal to internal jobs. Very roughly (i.e., up to one-step transitions), (5) is related to (4) as follows

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

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 [15], where *resumptions* from a domain Y satisfying

$$Y \quad = \quad D_\perp \to Pow(D_\perp + (D_\perp \times Y))$$

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

$$X \quad = \quad Pow((D \times (X + \{\surd\})) \times (D \times (X + \{\surd\})))\ . \tag{6}$$

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

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

following the associations

$$\begin{aligned}D \to Pow(D + (D \times Y)) \quad &\approx \quad (D \times \{\surd\}) \to Pow((D \times \{\surd\}) + (D \times Y)) \\ &\approx \quad (D \times \{\surd\}) \to Pow(D \times (Y + \{\surd\}))\ .\end{aligned}$$

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 essentially going back (at least) to Milner [12].

Remark. To confuse the symbol $\surd$ 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 $S = D \times (P_1 + \{\surd\})$. (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 $\surd$ is *not* a program, but is an indicator that the bag is empty. The identification of bags with elements of $P_1 + \{\surd\}$ 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 $\|$, 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

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

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

(1) becomes $\exists \bar{a}\ \ (d, \surd)\ [p, \bar{a}\rangle (d', \surd)$
(2) becomes $\forall d\ \exists d'\ \ (d, \surd)\ [p, a\rangle\ (d', p')$
(3) becomes $(d, \surd)\ [p, a\rangle\ (d', p')$
(4) becomes $\exists a\ \ (d, \surd)\ [p, a\rangle\ (d', p')$
(5) becomes $\exists \bar{a}\ \ (d_1, p_1)\ [p, \bar{a}\rangle\ (d_2, p_2)$.

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 $d, p, \theta, a, \bar{a}, \ldots$ are understood as schematic variables ranging over D, P_1, $\Theta = P_1 \cup \{\surd\}$, A, A^*, ..., 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

$$(d, a, d') \quad \frac{}{(d, \surd)\ [a, a\rangle\ (d', \surd)}$$

and close these transitions under

$$(;)\quad \frac{(d, \surd)\ [p_1, \bar{a}\rangle\ (d_1, \surd) \qquad (d_1, \surd)\ [p_2, \bar{b}\rangle\ (d', \theta)}{(d, \surd)\ [p_1; p_2, \bar{a}\bar{b}\rangle\ (d', \theta)}$$

$$(+\mathrm{r})\quad \frac{(d, \surd)\ [p_1, \bar{a}\rangle\ (d', \theta)}{(d, \surd)\ [p_1 + p_2, \bar{a}\rangle\ (d', \theta)} \qquad (\mathrm{l}+)\quad \frac{(d, \surd)\ [p_2, \bar{b}\rangle\ (d', \theta)}{(d, \surd)\ [p_1 + p_2, \bar{b}\rangle\ (d', \theta)}$$

$$(*)\quad \frac{(d, \surd)\ [skip + p; p^*, \bar{a}\rangle\ (d', \theta)}{(d, \surd)\ [p^*, \bar{a}\rangle\ (d', \theta)} \qquad (\|i)\quad \frac{(d, \surd)\ [p, \bar{a}\rangle\ (d', \surd)}{(d, p')\ [p, \bar{a}\rangle\ (d', p')}$$

$$(\|\text{-r})\quad \frac{(d, \surd)\ [p, \bar{a}\rangle\ (d', p_1)}{(d, p_2)\ [p, \bar{a}\rangle\ (d', p_1 \| p_2)} \qquad (\text{l-}\|)\quad \frac{(d, \surd)\ [p, \bar{a}\rangle\ (d', p_1)}{(d, p_2)\ [p, \bar{a}\rangle\ (d', p_2 \| p_1)}$$

$$(\text{sym})\quad \frac{(d, p_1)\ [p_2, \bar{a}\rangle\ (d', \theta)}{(d, p_2)\ [p_1, \bar{a}\rangle\ (d', \theta)} \qquad (\text{shift})\quad \frac{(d, p_1)\ [p_2, \bar{a}\rangle\ (d', \theta)}{(d, \surd)\ [p_1 \| p_2, \bar{a}\rangle\ (d', \theta)}$$

$$(\text{trans})\quad \frac{(d, \theta)\ [p, \bar{a}\rangle\ (d_1, p_1) \qquad (d_1, \surd)\ [p_1, \bar{b}\rangle\ (d', \theta')}{(d, \theta)\ [p, \bar{a}\bar{b}\rangle\ (d', \theta')}$$

$$(;i)\ \frac{(d,\surd)\ [p_1,\bar{a}\rangle\ (d',p)}{(d,\surd)\ [p_1;p_2,\bar{a}\rangle\ (d',p;p_2)} \qquad (;i')\ \frac{(d,\surd)\ [p_1,\bar{a}\rangle\ (d',\surd)}{(d,\surd)\ [p_1;p_2,\bar{a}\rangle\ (d',p_2)}\ .$$

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 $\|$ 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 $(d,p')\ [p\rangle\ (d_2,p_1\|p_2)$ from $(d,\surd)\ [p\rangle\ (d_1,p_1)$ and $(d_1,\surd)\ [p'\rangle\ (d_2,p_2)$ should be sufficient to make the point

$$\frac{\dfrac{(d,\surd)\ [p\rangle\ (d_1,p_1)}{(d,p')\ [p\rangle\ (d_1,p_1\|p')} \qquad \dfrac{\dfrac{(d_1,\surd)\ [p'\rangle\ (d_2,p_2)}{(d_1,p_1)\ [p'\rangle\ (d_2,p_1\|p_2)}}{(d_1,\surd)\ [p_1\|p'\rangle\ (d_2,p_1\|p_2)}}{(d,p')\ [p\rangle\ (d_2,p_1\|p_2)}$$

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

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

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

Observe that whenever $(d_1,p_1)\ [p,a_1\ldots a_n\rangle\ (d_2,p_2)$ is derivable, then $d_1\ I_{a_1}\circ\cdots\circ I_{a_n}\ d_2$. 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 begining 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

$$\begin{aligned}
[\![\cdot]\!]_0 &= \{(d,p,d')\mid (d,\surd)\ [p,\bar{a}\rangle\ (d',\surd) \text{ is derivable from axioms } (d_0,a_0,d_0') \\
&\qquad\qquad \text{by (;), (+r), (l+), and } (*), \text{ for some } \bar{a}\} \\
\rightarrow_3 &= \{((p,d),a,(\theta',d'))\mid (d,\surd)\overset{p,a}{\Longrightarrow}(d',\theta')\} \\
\rightarrow_4 &= \{(p,(d,d'),\theta')\mid (d,\surd)\overset{p,a}{\Longrightarrow}(d',\theta') \text{ for some } a\} \\
\rightarrow_{4.5} &= \{(p,(d,d'),\theta')\mid (d,\surd)\overset{p,\bar{a}}{\Longrightarrow}(d',\theta') \text{ for some } \bar{a}\} \\
\rightarrow_5 &= \{((d,\theta),p,(d',\theta'))\mid (d,\theta)\overset{p,\bar{a}}{\Longrightarrow}(d',\theta') \text{ for some } \bar{a}\}\ .
\end{aligned}$$

We show next that $\Longrightarrow$ extends $[\![\cdot]\!]_0$ "conservatively" in the sense that for every $p \in P_0$ and all $d, d' \in D$, if $(d, \surd) \stackrel{p,\bar{a}}{\Longrightarrow} (d', \surd)$ for some $\bar{a}$, then $d[\![p]\!]_0 d'$. 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

$$\begin{aligned} [\![\cdot]\!]_0 &= \{(d,p,d') \mid (d,\surd)\ [p\rangle\ (d',\surd) \text{ is derivable from } \Gamma_0\} \\ \rightarrow_5 &= \{((d,\theta),p,(d',\theta')) \mid (d,\theta)\ [p\rangle\ (d',\theta') \text{ is derivable from } \Gamma_1\}\ . \end{aligned}$$

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

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

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 $(d,\surd)\ [p\rangle\ (d',\theta)$, it is the case that $\theta \in P_0 \cup \{\surd\}$ 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 $(d,\surd)\ [p\rangle\ (d',\theta)$, let $\varphi(D)$ be the assertion

> if $\theta = \surd$, then Γ_0 proves that $(d,\surd)\ [p'\rangle\ (d',\surd)$,
> else if $\theta = p'$ and Γ_0 proves that $(d',\surd)\ [p'\rangle\ (d'',\surd)$, then Γ_0 proves that $(d,\surd)\ [p\rangle\ (d'',\surd)$.

Now, induct on the length of D, noting from (†) that $\theta \in P_0 \cup \{\surd\}$, 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. For lack of space, this is left to the reader. ⊣

Having established Theorem 1, note 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 $\langle P, D \times \Theta, [\cdot\rangle_\Gamma\rangle$ (with $\Theta = P + \{\surd\}$) whose transition relation is given by some rule set Γ (such as Γ_0 or Γ_1) as follows

$$(d,\theta)\ [p\rangle_\Gamma\ (d',\theta') \quad \text{iff} \quad (d,\theta)\ [p\rangle\ (d',\theta') \text{ follows from } \Gamma\ .$$

Notice that "$(d,\theta)\ [p\rangle\ (d',\theta')$" 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

$$(d,\cdot)\ [\cdot\rangle\ (d',\cdot)\ ,\ (d,\surd)\ [\cdot\rangle\ (d',\cdot)\ ,\ (d,\cdot)\ [\cdot\rangle\ (d',\surd)\ ,\ (d,\surd)\ [\cdot\rangle\ (d',\surd)$$

for every d and $d' \in D$. The holes $\cdot$ are to be filled by programs in P. Now, a "semantics" for the set P of programs is at the very least a map f with domain P. 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 $(d,\theta_0)\ [p\rangle_\Gamma\ (d',\theta_0')$ must hold in f. Without saying anything about the co-domain of f, let us focus on the "equality" relation $\{(p,p') \in P \times P \mid f(p) = f(p')\}$ on P 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 $[\cdot\rangle_\Gamma$ under equality

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

where, for notational convenience, f has been extended to Θ by setting $f(\surd) = \surd$. Now, a minimal[2] 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 $\theta, \theta' \in \Theta$,

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

In other words, if $[f]_\Gamma$ is defined as the map from P given by

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

then

$$(\star)\ \ (\forall p,p' \in P)\ f(p) = f(p') \text{ implies } [f]_\Gamma(p) = [f]_\Gamma(p')\ ,$$

or, equivalently,

$(\star)'$ for all $p, p', d_1, d_2, \theta_1, \theta_2$ such that $f(p) = f(p')$ and $(d_1,\theta_1)\ [p\rangle_\Gamma\ (d_2,\theta_2)$, there exist θ_1' and θ_2' such that $f(\theta_1') = f(\theta_1)$, $f(\theta_2') = f(\theta_2)$ and $(d_1,\theta_1')\ [p'\rangle_\Gamma\ (d_2,\theta_2')$.

[2] 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 $(d,\cdot)\ [\cdot\rangle\ (d',\cdot)$, $(d,\surd)\ [\cdot\rangle\ (d',\cdot)$, $(d,\cdot)\ [\cdot\rangle\ (d',\surd)$, and $(d,\surd)[\cdot\rangle\ (d',\surd)$ 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.

A second condition to impose on f is

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

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 $\surd \notin P$ (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, is $[f]_\Gamma$.

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 $f = [f]_\Gamma$. 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 $f = [f]_\Gamma$. (Readers familiar with Aczel [1] can see this by noting that for every $p \in P$,

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

and that $\surd$ 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 $[\![\cdot]\!]_\Gamma$. One way to see the importance of $[\![\cdot]\!]_\Gamma$ is through a little category theory.

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

$$\alpha^f_\Gamma(f(p)) \ = \ [f]_\Gamma(p)$$

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

$$\alpha^g_\Gamma(\varphi f p) \ = \ \{((d, \varphi f\theta), (d', \varphi f\theta')) \mid (d,\theta)\ [p\rangle_\Gamma\ (d',\theta')\}\ .$$

This equation can be pictured as follows

$$\begin{array}{ccccc} P & \xrightarrow{\ f\ } & f''P & \xrightarrow{\ \alpha^f_\Gamma\ } & F(f''P) \\ & & \downarrow{\scriptstyle \varphi} & & \downarrow{\scriptstyle F(\varphi)} \\ P & \xrightarrow{\ g\ } & g''P & \xrightarrow{\ \alpha^g_\Gamma\ } & F(g''P) \end{array}$$

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

$$F(X) \quad = \quad Pow((D \times (X + \{\surd\})) \times (D \times (X + \{\surd\})))$$

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

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

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

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 $\mathbf{C}_\Gamma$, the first of which expresses that the input-output behavior prescribed by Γ is respected.

Theorem 2.

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

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

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 $\sim_\Gamma$ above with the help of AFA, it is possible to forget $[\![\cdot]\!]_\Gamma$ and AFA, and "simply" show that the function sending $\surd$ to $\surd$ and p to $\{p' \in P \mid p \sim_\Gamma p'\}$ is final in $\mathbf{C}_\Gamma$. 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 [10]. One other point that Appendix A attends to is that Γ is indeed sound for $[\![\cdot]\!]_\Gamma$ (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

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

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 $(d_1, \theta_1) \xrightarrow{p}_5 (d_2, \theta_2)$, there exist θ_1' and θ_2' such that $f(\theta_1') = f(\theta_1)$, $f(\theta_2') = f(\theta_2)$ and $(d_1, \theta_1') \xrightarrow{p'}_5 (d_2, \theta_2')$,

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 [13] is given as follows (modified slightly to take $\surd$ into account). Define a *bisimulation* on a transition system $\langle L, S, \rightarrow \rangle$ to be a relation $R \subseteq S \times S$ such that whenever sRs', then for every $l \in L$,

$$(\forall t \xleftarrow{l} s)\ (\exists t' \xleftarrow{l} s')\ tRt' \quad \text{and} \quad (\forall t' \xleftarrow{l} s')\ (\exists t \xleftarrow{l} s)\ tRt' \ ,$$

and $s = \surd$ iff $s' = \surd$. The $\forall\exists$-conjuncts 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 $\leftrightarroweq$ on $\langle L, S, \rightarrow \rangle$ by

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

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 [10] (see also Rutten [16]) that the bisimilarity predicate $\leftrightarroweq_4$ for Example 4 is a congruence. This property is preserved by the addition of the rule (trans); that is, the predicate $\leftrightarroweq_{4.5}$ for $\langle D \times D, \Theta, \rightarrow_{4.5} \rangle$ is a congruence.

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

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

Adding the rule (trans) to Example 4 represents a real change in that, for instance, $a + skip; a \ \sim \ skip; a$ (although $skip; skip \not\sim skip$) while Example 4 differentiates between these when $I_a \neq I_{skip}$. 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. *$\leftrightarrows_4$ is finer than $\sim$, and, in general, strictly so.*

Proof. Following the previous proof, define a function f with domain Θ by setting $f(\surd) = \surd$ and for $p \in P_1$, $f(p) = \{\theta \mid p \leftrightarrows_4 \theta\}$. Next, to see that f is Γ_1-consistent, it suffices to show that

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

The assertion (†) holds because (i) the rule (;) can be replaced by (trans) and (;i'), 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

$$\dfrac{\dfrac{(d,\theta)\ [skip + p; p^*\rangle\ (d_1,p') \qquad\qquad (d_1,\surd)\ [p'\rangle\ (d',\theta')}{(d,\theta)\ [skip + p; p^*\rangle\ (d',\theta')}}{(d,\theta)\ [p^*\rangle\ (d',\theta')}$$

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

$$\dfrac{\dfrac{(d,\theta)\ [skip + p; p^*\rangle\ (d_1,p')}{(d,\theta)\ [p^*\rangle\ (d_1,p')} \qquad\qquad (d_1,\surd)\ [p'\rangle\ (d',\theta')}{(d,\theta)\ [p^*\rangle\ (d',\theta')}$$

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

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 [9] to be a relation $R \subseteq (P \times D) \times (P \times D)$ such that whenever $(p,d)R(p',d)$, then for every $a \in A$,

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

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

$p \leftrightarrows_3 p'$ iff for all $d \in D$ there is a global D-bisimulation R such that $(p,d)R(p',d)$.

Theorem 5. *$\leftrightarrows_3$ is finer than $\leftrightarrows_4$, and, in general, strictly so.*

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 $(p_0',d') \xleftarrow{a}_3 (p_0,d)$, R relates (p',d'') to (p_0',d'') for every d''. But in that case, $p_0 \xrightarrow{d,d'}_4 p_0'$. Hence, $p \leftrightarrows_3 p'$ implies $p \leftrightarrows_4 p'$.

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 $I_{a_1} \cup I_{a_2} = I_{a_3} \cup I_{a_4}$). ⊣

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

$$(\text{at}) \quad \frac{(d, \surd)\ [p, \overline{a}\rangle\ (d', \surd)}{(d, \surd)\ [[p], [p]\rangle\ (d', \surd)}\ ,$$

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. It would be interesting to "eliminate" so-called τ- or silent steps by suitable applications of the atomization construct.) The congruence and soundness results described in Appendix A 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 would seem to be interesting topics to investigate. 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.

A final point (suggested by the reference in section 3 to the set theory in Aczel [1]) is that the step from syntactic presentations of transition systems to semantic (extensional) notions of identity might be analyzed against a generalization of ordinary (numerical) recursion theory to a computational theory for sets. In particular, the set-theoretic recursion theory 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] [8]).

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. Between programs and processes: absoluteness and open ended-ness. Technical Report IAM 92-011, Institut für Informatik, Universität Bern, 1992.
9. 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.
10. 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.
11. David Harel. Dynamic logic. In Gabbay et al, editor, *Handbook of Philosophical Logic, Volume 2*. D. Reidel, 1984.
12. Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25, 1983.
13. David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI Conference*, LNCS 104. Springer-Verlag, Berlin, 1981.
14. David Peleg. Concurrent dynamic logic. *J. Assoc. Computing Machinery*, 34(2), 1987.
15. Gordon D. Plotkin. A powerdomain construction. *SIAM J. Comput.*, 5(3), 1976.
16. 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 $\mathbf{C}_\Gamma$-object id, but is most interesting for the final $\mathbf{C}_\Gamma$-objects. The problem of interpreting the program constructs $;, +, ^*$ and $\|$ 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 $\bullet$ — 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 [10], 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 $\mathbf{C}_\Gamma$-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 $[\cdot]_\Gamma$ is, however, sufficient for a wide collection of rules, as will be shown shortly.

Given a set X of program variables and a family $\mathcal{F} \supseteq A$ of function symbols of various arities including 0 (e.g., ;, *, $a \in A$), let $\mathcal{F}(X)$ be the resulting collection of program terms with program variables in X. (Thus, $P_1 = (A \cup \{+, ;, ^*, \|\})(\emptyset)$.) Define R to be the following relation on $\mathcal{F}(\emptyset)$

$$\{(t[p_1, \ldots, p_n], t[p'_1, \ldots, p'_n]) \mid n < \omega,\ t(x_1, \ldots, x_n) \in \mathcal{F}(\{x_1, \ldots, x_n\}) \text{ and } p_1 \sim_\Gamma p'_1\ ,\ \cdots\ , p_n \sim_\Gamma p'_n\} .$$

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 $(d_1, \theta_1)\ [p\rangle_\Gamma\ (d_2, \theta_2)$,

$$\exists \theta'_1, \theta'_2 \text{ s.t. } (\theta_1, \theta'_1) \in R \cup \{(\surd, \surd)\}\ ,\ (\theta_2, \theta'_2) \in R \cup \{(\surd, \surd)\} \text{ and } (d_1, \theta'_1)\ [p'\rangle_\Gamma\ (d_2, \theta'_2) .$$

Lemma A (AFA). *For every nice rule set Γ, $\sim_\Gamma$ is a congruence with respect to every function symbol in $\mathcal{F}$.*

Proof. Since Γ is nice, it follows that the map f sending $\surd$ to $\surd$ and every $p \in \mathcal{F}(\emptyset)$ to $\{p' \mid pRp'\}$ is a $\mathbf{C}_\Gamma$-object. But by Theorem 2, $[\![\cdot]\!]_\Gamma$ is final, whence R is $\sim_\Gamma$, 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. $(\theta_1, \theta'_1) \in R \cup \{(\surd, \surd)\}$, if $(d_1, \theta_1)\ [p\rangle_\Gamma\ (d_2, \theta_2)$ then $\exists \theta'_2$ s.t. $(\theta_2, \theta'_2) \in R \cup \{(\surd, \surd)\}$ and $(d_1, \theta'_1)\ [p'\rangle_\Gamma\ (d_2, \theta'_2)$.

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 $\mathcal{F}(X)$ *primitive* if it is either (a constant denoting) an element of A or a program variable ($\in X$). For $\hat{\theta} \in \mathcal{F}(X) \cup \{\surd\}$, let $Var(\hat{\theta})$ be the set of variables in X occurring in $\hat{\theta}$. Now, consider a rule of the form

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

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 $\theta' \in \mathcal{F}(X) \cup \{\surd\}$,
for $i < n$, $p_i \in \mathcal{F}(X)$,
(c1) θ and every θ'_i $(i < n)$ are primitive program terms or $\surd$,
(c2) $Var(\theta) \cap Var(p) = \emptyset$,
(c3) $Var(\theta'_i) \cap Var(\theta'_j) = \emptyset$ for $i < j < n$,
(c4) $Var(p_i) \cap Var(\theta'_j) = Var(\theta_i) \cap Var(\theta'_j) = \emptyset$ for $i \leq j < n$,
(c5) $Var(p) \cap Var(\theta'_i) = Var(\theta) \cap Var(\theta'_i) = \emptyset$ for $i < n$, and
(c6) p is either a primitive program term or $f(x_1, \ldots, x_n)$ for some n-ary $f \in \mathcal{F}$.

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 $\Theta = P_1 \cup \{\surd\}$.

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 $(d_1,\theta_1)\ [p\rangle_\Gamma\ (d_2,\theta_2)$ and $(\theta_1,\theta_1') \in R\cup\{(\surd,\surd)\}$, then there is a θ_2' such that $(\theta_2,\theta_2') \in R\cup\{(\surd,\surd)\}$ and $(d_1,\theta_1')\ [p\rangle_\Gamma\ (d_2,\theta_2')$.

A proof of (†) proceeds by induction on the length of Γ-derivations. Consider the last rule (r) of a shortest Γ-derivation of $(d_1,\theta_1)\ [p\rangle\ (d_2,\theta_2)$. 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 $(\theta_2,\theta_2') \in R\cup\{(\surd,\surd)\}$ and the conclusion of (r) can be instantiated by $(d_1,\theta_1')\ [p\rangle\ (d_2,\theta_2')$.

Now, suppose $(p,p') \in R$, $\theta_1,\theta_2 \in \Theta$, $d_1,d_2 \in D$, $(\theta_1,\theta_1') \in R\cup\{(\surd,\surd)\}$, and $(d_1,\theta_1)\ [p\rangle_\Gamma\ (d_2,\theta_2)$. If $p \sim_\Gamma p'$ then appeal to (†) above. Otherwise, adapt the induction argument for (†) using (c6) to replace p by p'. ⊣

Lemmas A and B yield

Theorem C (AFA). *If Γ is a set of rules with the form of (r), then $\sim_\Gamma$ is a congruence with respect to every function symbol in $\mathcal{F}$.*

Note that the theorem applies to Γ_1, and the connection with the general congruence results of Groote and Vaandrager [10] is made above for the particular case of $;,+,\|$ and *, with interleaving 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 Γ is a set of rules with the form of (r), then Γ is sound for $[\cdot]_\Gamma$.*

Appendix B: living with 'programs as relations'

The "parallel" construct $\|$ 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 $\|_H$, described in the usual transition system format of Example 2 (without data) by the rules

$$(\alpha)\quad \frac{p_1 \xrightarrow{a} p_1' \qquad p_2 \xrightarrow{a} p_2'}{p_1\|_H p_2 \xrightarrow{a} p_1'\|_H p_2'}\ a\in H \qquad\qquad (\beta)\quad \frac{p_1 \xrightarrow{a} p_1'}{p_1\|_H p_2 \xrightarrow{a} p_1'\|_H p_2}\ a\notin H\ .$$

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

$$\frac{(d,\surd)\ [p_1,a\rangle\ (d',p_1')}{(d,\surd)\ [p_1\|_H p_2,a\rangle\ (d',p_1'\|_H p_2)}\ a\notin H$$

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

$$\frac{(d_1,\surd)\ [p_1,a\rangle\ (d_1',p_1') \qquad (d_2,\surd)\ [p_2,a\rangle\ (d_2',p_2')}{(i(d_1,d_1',d_2,d_2'),\surd)\ [p_1\|_H p_2,a\rangle\ (j(d_1,d_1',d_2,d_2'),p_1'\|_H p_2')}\ a\in H\ .$$

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 $\|$ 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

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

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

$$(\alpha)'\ \ \frac{(d_1, \surd)\ [p_1\rangle\ (d_1', p_1') \qquad (d_2, \surd)\ [p_2\rangle\ (d_2', p_2')}{(d, \surd)\ [p_1 \|_H p_2\rangle\ (d', p_1' \|_H p_2')}\ \ S_H(d_1, d_1', d_2, d_2', d, d')$$

where

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

Similarly, for (β), take

$$(\beta)'\ \ \frac{(d, \surd)\ [p_1\rangle\ (d', p_1')}{(d, \surd)\ [p_1 \|_H p_2\rangle\ (d', p_1' \|_H p_2)}\ \ d'(\hat{s}) \notin H\ .$$

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 $S_{\&}$

$$(\alpha)''\ \ \frac{(d_1, \surd)\ [p_1\rangle\ (d_1', p_1') \qquad (d_2, \surd)\ [p_2\rangle\ (d_2', p_2')}{(d, \surd)\ [p_1 \& p_2\rangle\ (d', [p_1', p_2']_{\&})}\ \ S_{\&}(d_1, d_1', d_2, d_2', d, d')\ .$$

(In Groote and Ponse [9], for example, note that the construct $\mid$ is continued by $\|$.) The possibilites 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 $S = D \times (P + \{\surd\})$ would be silly.