A Branching Time Semantics for the Ada*Rendezvous Mechanism †

Boumediene Belkhouche
Computer Science Dept.
Tulane University
New Orleans, LA

R.Raymond Lang
Computer Science Dept.
Xavier University
New Orleans, LA

Chung Wa Ng
Computer Science Dept.
Moorehouse College
Atlanta, GA

Abstract

Branching-time semantics based on domains built upon tree structures have been proposed to model concurrent processes. However, the resulting models imposed severe restrictions to ensure monotonicity and compositionality. To address these issues, we construct a semantic domain without sacrificing these two properties. We also provide a simple and faithful semantics of the Ada rendezvous mechanism.

1 Introduction

We propose a tree-based semantic domain to capture the denotational semantics of a uniform language supporting the rendezvous mechanism. The domain we construct provides a means of distinguishing processes which share the same language but which differ as to their respective choices points. In other words, we develop what is commonly known as a branching time semantics [2].

A detailed presentation of the language is given in the next section. In Section 3, we explain the structure of the domain. This will form the basis of our model. Section 4 gives definitions of the semantic function and the finite elements of the signature. Section 5 proves monotonicity and continuity in order to give a meaning to infinite processes.

2 The Language $\mathcal{L}$

The language we consider supports nondeterminism and synchronization based on the rendezvous. In order to focus more closely on the semantic issues involved in our language, the actions are left uninterpreted. Thus, our uniform language $\mathcal{L}$ is defined as a successive refinement of three sublanguages ($\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$) described by the following production rules:

$\mathcal{L}_1$

$$\mathtt{p} ::= \mathtt{skip} \mid \mathtt{stop} \mid \mathtt{a} \mid \mathtt{p};\mathtt{p} \mid \mathtt{p}+\mathtt{p}$$

$\mathcal{L}_2$

$$\mathtt{p} ::= \mathtt{skip} \mid \mathtt{stop} \mid \mathtt{a} \mid \mathtt{p};\mathtt{p} \mid \mathtt{p}+\mathtt{p} \mid \mathtt{p} \parallel \mathtt{p}$$

$\mathcal{L}_3$

$$\begin{aligned} \mathtt{p} \; ::= \; & \mathtt{skip} \mid \mathtt{stop} \mid \mathtt{a} \mid \mathtt{p};\mathtt{p} \mid \mathtt{p}+\mathtt{p} \mid \mathtt{p} \parallel \mathtt{p} \\ & \mid \sigma \mid \overline{\sigma} : \mathtt{p} \mid \mu \mathtt{x}.\mathtt{p} \end{aligned}$$

We now give a brief informal description of each of the language constructs.

1. the nullary action `skip`. The sole result of this process is immediate normal termination.
2. the nullary action `stop`. The sole result of this process is immediate deadlock.
3. an elementary action $\mathtt{a} \in \mathtt{Act}$. Actions of this kind are left uninterpreted in the sense that they are given no specification either in concrete terms or in terms of state transforming functions. The set `Act` of actions is assumed countable.
4. the sequential composition of two processes, signified by `p;p`.
5. the nondeterministic choice between two processes: `p + p`.
6. the parallel composition of two processes: $\mathtt{p} \parallel \mathtt{p}$.
7. σ denotes a request to rendezvous with another process which has entry point $\overline{\sigma}$. The action σ represents an *entry call.*
8. $\overline{\sigma} : \mathtt{p}$, signals a readiness to accept a rendezvous and execute critical region `p`. The action $\overline{\sigma} : \mathtt{p}$ represents an *entry accept.*
9. $\mu X.p$ expresses infinite repetition of the atomic actions of `p`.

*Ada is a registered trade mark of the US DoD (AJPO)
†Published in HICSS-29, January 1996

3 The Semantic Domain

Two major approaches modeling processes as tree structures have been proposed. Metric spaces are used to build semantic domains in [3, 2]. One serious drawback of this approach is that sequential composition is restricted to processes whose trees are full trees (i.e., all branches are of the same depth). Such a restriction was implicitly motivated by the need to preserve monotonicity of the sequential composition operator. Indeed, for trees that are not full, sequential composition is not monotonic. Acceptance trees are used in [4, 5]. This approach results in a very complicated nondeterministic composition, making it non-compositional. In departure from this, we allow our domain to contain trees with branches of arbitrary depth without sacrificing monotonicity or compositionality.

3.1 Tree Representation

A process (i.e., an expression of the language) is represented as a tree that is encoded linearly as a set. Such a set is defined structurally in the following way.

(1) `skip` is encoded as $\{\epsilon\}$. Pictorially, a $\bullet$ labelled $SKIP$ is used. The $\bullet$ stands for a closed tree node.

(2) `stop` is encoded as $\{\delta\}$. Pictorially, a $\circ$ labelled $STOP$ is used. This is called an open node.

(3) `a` is encoded as $\{\langle a, SKIP\rangle\}$. Pictorially, a rooted tree with a single branch labelled a and a leaf node labelled $SKIP$ is used.

(4) `a;p` is encoded as $\{\langle a, t\rangle\}$. Pictorially, if t is the tree representing the expression p, and t' the branch representing a, then the new tree for `a;p` is formed by superimposing the leaf node of t' and the root node of t. The resulting node is not labelled. The new root node is the root node of t'.

(5) `p+q` is encoded as $\{p', q'\}$, where p' and q' are the encoding for `p` and `q`, respectively. Pictorially, the roots of the trees corresponding to `p` and `q` are superimposed to form a single root.

(6) $\mu X.p$ is encoded as $\{\ s^n\!:\ s$ is the tree encoding for `p` and $n \in \omega\ \}$.

Therefore, each process expression is a set. We will use $SKIP$ to stand for the set $\{\epsilon\}$, and $STOP$ to stand for the set $\{\delta\}$.

We construct the semantic domain for our language as the limit of a series of sets P_n. Informally, each P_n contains all the possible processes in which the longest branches are of length n. Formally, let P_0 be defined as $\{\emptyset, SKIP, STOP\}$. These eleemnts are (1) $\emptyset$, the process about which we have no information (bottom); (2) $SKIP = \{\epsilon\}$, the process which does nothing and terminates normally; and (3) $STOP = \{\delta\}$, the process which does nothing and halts abnormally. Then, for every $n > 0$, let P_n be defined as:

$$P_n = \mathcal{P}((Act \times (P_{n-1} - \{\emptyset\})) \cup \{SKIP, STOP\})$$

where $\mathcal{P}(\cdot)$ signifies the power set. In building each set P_n, we remove $\emptyset$ from P_{n-1} and take the cross product of this with Act. The cross-product forms a set of pairs. We then take the union of this set of pairs with the set $\{\epsilon, \delta\}$. P_n is defined as the power set of this union. First of all, we prevent pairs of the form $\langle a, \emptyset\rangle$ from arising by removing the empty set from P_{n-1} before forming the cross product with Act. By forming the cross product of P_{n-1} and Act at each step, the creation of a new P_n adds one to the length of each path of each tree in P_{n-1}. However, if we do not add the elements ϵ and δ to *each* cross product, P_n will contain only processes in which each branch has a depth of n. We would not obtain processes with branches of varying depths; but it is precisely these types of processes that we explicitly wish to include in our domain. For example, with $Act = \{a, b\}$:

$$\begin{aligned} P_1 &= \mathcal{P}((Act \times (P_0 - \{\emptyset\})) \cup \{SKIP, STOP\}) \\ &= \mathcal{P}(\{a, b\} \times \{SKIP, STOP\}) \cup \{SKIP, STOP\}) \\ &= \mathcal{P}(\{SKIP, STOP, \langle a, SKIP\rangle, \langle b, SKIP\rangle, \\ &\qquad \langle a, STOP\rangle, \langle b, STOP\rangle\}) \end{aligned}$$

The definition of P_n that we have given may appear contrived at first glance. However, on closer inspection, the reader will find that we have assigned an interpretation to sets containing pair(s) of the form $\langle a, \emptyset\rangle$. Such sets arise if P_n is given the much simpler definition $\mathcal{P}(Act \times P_{n-1})$, which is more common in the literature [2]. In our formulation, sets such as $\{\langle a, \emptyset\rangle\}$ do not arise; but the three processes $\{\langle a, SKIP\rangle\}$, $\{\langle a, STOP\rangle\}$, and $\{\langle a, SKIP\rangle, \langle a, STOP\rangle\}$ do appear. Informally, if processes containing pairs of the form $\langle a, \emptyset\rangle$ are regarded as abbreviations for all the processes containing either $\langle a, SKIP\rangle$, $\langle a, STOP\rangle$, or both in the corresponding position, our formulation of the domain simply forces the expansion of this abbreviation.

A non-empty set is interpreted as non-deterministic choice among its elements. Each pair, then, represents a primitive action followed by a nondeterministic choice among the elements of some set. The nesting of sets which occurs in every P_n for $n \geq 2$ indicates that the domain we are building is tree structured. Intuitively, the pairs map to branches and the non-empty sets to nodes. This is important as we intend to provide a branching time semantics which will differentiate between $(a; b) + (a; c)$ and $a; (b + c)$.

P_n being a power set, it is a complete algebraic lattice, and thus a domain. The algebraicity of P_n allows us to define infinite processes in terms of finite ones [1]. At this point, certain features of the domain are already apparent:

1. For every set $S \in P_n$, $e \in S$ is either (1) ϵ, (2) δ, or (3) a pair of the form $\langle a, S \rangle$ where $a \in Act$ and $S \in P_m$ for some $m \leq n$. Later, we will prove that this feature characterizes our domain in the sense that it is necessary and sufficient for all elements of the domain.
2. Each element of a given P_n is unique (this follows immediately from the set operations used to construct each P_n).
3. For all $n > 0$, $P_{n-1} \subseteq P_n$
4. The elements of each P_n correspond to all the possible processes which can be constructed from the atomic actions of *Act* and in which all branches have depth $\leq n$. This includes both all possible sequences of actions and all possible choice points for sequences of actions which share a common prefix.

Because of Property 4, processes which produce the same language but differ in their respective choice points will have distinct representations. For example, the processes $p = a; (a + b)$ and $q = (a; a) + (a; b)$ appear separately in P_2 as

$$\{\langle a, \{\langle a, SKIP \rangle, \langle b, SKIP \rangle\}\rangle\}$$

and

$$\{\langle a, \{\langle a, SKIP \rangle\}\rangle \ , \ \langle a, \{\langle b, SKIP \rangle\}\rangle\}$$

Furthermore, because

$$\{\langle a, SKIP \rangle, \langle b, SKIP \rangle\} = \{\langle b, SKIP \rangle, \langle a, SKIP \rangle\}$$

we can predict that the non-deterministic choice operator will be both commutative and associative.

The semantic domain is the limit of P_n as n approaches infinity:

$$P_\omega \stackrel{\text{def}}{=} \lim_{n \to \infty} P_n$$

Property 1 There is a property of the elements of P_ω which is necessary and sufficient for a set to be included in P_ω: $\forall S, S \in P_\omega$ iff $\forall e \in S$ one of the following holds:

(i) $e = SKIP$

(ii) $e = STOP$

(iii) $e = \langle a, S' \rangle$ for some $a \in Act$ and some $S' \in (P_\omega - \{\emptyset\})$.

In order to prove that Property 1 characterizes P_ω, we must first prove the following lemma:

Lemma 1 $\forall S, T \in P_\omega, \ S \cup T \in P_\omega$.

Proof Let S and T be arbitrary elements of P_ω. Then since each element of P_ω is an element of P_n for some n, we may infer that $S \in P_x$ and $T \in P_y$, for some x, y. There are three cases to consider:

(i) $x = y$, then S and T are elements of the same power set, namely P_x, and $S \cup T$ must also be $\in P_x$, since P_x is a power set. Therefore $(S \cup T) \in P_\omega$.

(ii) $x > y$, then $P_y \subseteq P_x$ since for all n, $P_n \subseteq P_{n+1}$. This implies that $T \in P_x$. Since S and T are both $\in P_x$, $(S \cup T) \in P_\omega$.

(iii) $x < y$, this case is identical to case 2. $\square$

We now prove that Property 1, described above, holds for any element of P_ω and that any set S that possesses Property 1 is an element of P_ω.

Lemma 2 $\forall S, S \in P_\omega$ iff $\forall e \in S$, Property 1 holds.

Proof Let S be some element of P_ω

$\Rightarrow S \in P_n$, for some n. The proof is by induction on n.

Also, let S be a set such that Property 1 holds $\Rightarrow S \in P_\omega$.

$\Rightarrow$ $S \in P_n$, for some n

(1) if $n = 0 \Rightarrow S \in P_0 = \{\emptyset, SKIP, STOP\}$
and Property 1 holds.

(2) if $n > 0 \Rightarrow$
$P_n = \mathcal{P}((Act \times (P_{n-1} - \{\emptyset\})) \cup \{SKIP, STOP\})$
$\Rightarrow S \in \mathcal{P}((Act \times (P_{n-1} - \{\emptyset\})) \cup \{SKIP, STOP\})$
$\Rightarrow S \subseteq ((Act \times (P_{n-1} - \{\emptyset\})) \cup \{SKIP, STOP\})$
$\Rightarrow \forall e \in S$,
(i) $e = SKIP$, or
(ii) $e = \delta$, or
(iii) $e = \langle x, y \rangle$ for some $x \in Act$
and $y \in P_{n-1} - \{\emptyset\}$
then $P_{n-1} - \{\emptyset\} \subseteq P_\omega \Rightarrow y \in P_\omega$
$\Rightarrow$ Property 1 holds.

S has Property 1 $\Rightarrow S \in P_\omega$
By induction on the size of S where S_n denote S such that $|S| = n$,

Base Case: $|S| = 1, S = \{e\}$

(i) if $e = \epsilon$ then $S = SKIP \in P_\omega$
(ii) if $e = \delta$ then $S = STOP \in P_\omega$
(iii) if $e = \langle a, S' \rangle \in S$
for some $a \in Act$ and $S' \in (P_n - \{\emptyset\})$ for some n
$\Rightarrow e \in Act \times (P_n - \{\emptyset\})$
$\Rightarrow \{e\} \in \mathcal{P}(Act \times (P_n - \{\emptyset\})) \subseteq P_{n+1}$
$\Rightarrow S \in P_{n+1} \subseteq P_\omega$
$\Rightarrow S \in P_\omega$

Induction Step: $|S| = n$.
Assume that $S_n = \{e_1, e_2, \ldots, e_n\} \in P_\omega$
where $\forall\ 1 \leq i \leq n, e_i = \langle a, S' \rangle$
for some $a \in Act$ and $S' \in (P_m - \{\emptyset\})$ for some m
$S_{n+1} = \{e_1, e_2, \ldots, e_n, e_{n+1}\}$
where e_{n+1} satisfies one of the three subcases of Property 1
$S_{n+1} = S_n \cup \{e_{n+1}\}$
Since $S_n \in P_\omega$ by induction,
and$\{e_{n+1}\} \in P_\omega$ by the Base Case
$\Rightarrow S_{n+1} \in P_\omega$ by Lemma 1 □

In order to show that P_ω is a domain, we must (1) define an ordering on its elements; (2) show that there is a least element in P_ω; and (3) show that every ordered chain of elements has a least upper bound (*lub*).

3.2 Ordering

We define an ordering relation on processes, $\sqsubseteq$: $P_\omega \times P_\omega$. The relation $\sqsubseteq$ is defined as follows:

(i) $\emptyset \sqsubseteq p,\ \forall p \in P_\omega$

(ii) $SKIP \sqsubseteq p,\ \forall p \in (P_\omega - \{\emptyset, STOP\})$

(iii) $STOP \sqsubseteq p,\ \forall p \in (P_\omega - \{\emptyset, SKIP\})$

(iv) $\forall p, q \in P_\omega,\ p \sqsubseteq q$ iff $p \subseteq q$ or $p \subset' q$.

Where $\subset'$ is defined as follows. Assume $p = \{\langle x, T \rangle\}$ and $q = \{\langle y, R \rangle\}$. Then $p \subset' q$ iff

(i) $x = y$, and

(ii) $\forall T_i \in T\ \exists R_j \in R : \{T_i\} \sqsubseteq \{R_j\}$

This ordering conveys the notion that the tree p can be embedded from the root in the tree q (i.e., p is tree-like prefix of q). In order to demonstrate monotonicity for each of the two basic binary operators in the signature (+ , ;) we need to extend the ordering relation, $\sqsubseteq$: $P_\omega \times P_\omega$, to $\sqsubseteq'$: $(P_\omega \times P_\omega) \times (P_\omega \times P_\omega)$ such that:

$\forall p, q, p', q' \in P_\omega, (p, q) \sqsubseteq' (p', q')$ iff $p = p'$ and $q \sqsubseteq q'$

$(P_\omega, \sqsubseteq)$ is a partial order because

(i) $\forall p \in P_\omega, p \sqsubseteq p$; and

(ii) $\forall p, q \in P_\omega,\ p \sqsubseteq q$ and $q \sqsubseteq p \Rightarrow p = q$; and

(iii) $\forall p, q, r \in P_\omega, p \sqsubseteq q$ and $q \sqsubseteq r \Rightarrow p \sqsubseteq r$.

A *chain* of processes is a sequence $X = < x_i >_i$ such that $x_i \sqsubseteq x_{i+1}, i = 0, 1 \ldots$. The least upper bound (lub) of such a chain $X \subseteq P_\omega$ is that element $y \in P_\omega$ such that

(i) $\forall x \in X,\ x \sqsubseteq y$

(ii) $\forall z \in P_\omega,\ (\forall x \in X,\ x \sqsubseteq z \Rightarrow y \sqsubseteq z)$

Since P_ω is defined in terms of power set, P_ω is a lattice and every chain of processes $X = < x_i >_i \subseteq P_\omega$ has a least upper bound. Moreover, since $\emptyset \sqsubseteq p, \forall p \in P_\omega$, $\emptyset$ is the least element of P_ω. Therefore our domain, the triple $(P_\omega, \sqsubseteq, \emptyset)$, is a complete partial order. We now proceed to describe the mappings from T_Σ onto this domain.

4 Semantics of $\mathcal{L}$

Given this model, $\mathcal{L}$ is characterized by the following axioms:

(A1) `p;skip = p`

(A2) `skip;p = p`

(A3) `stop;p = stop`

(A4) `(p;q);r = p;(q;r)`

(A5) `p + p = p`

(A6) `p + q = q + p`

(A7) `(p + q) + r = p + (q + r)`

It should be noted that these axioms are indeed compatible with the informal semantics of the rendezvous mechanism.

4.1 Semantic Function

The semantic function $TR : \mathcal{L} \mapsto P_\omega$ maps an expression in the language into an element of the semantic domain corresponding to the tree depicting the process signified by that expression. TR is defined as follows:

(i) $TR[\texttt{skip}] = SKIP = \{\epsilon\}$

(ii) $TR[\texttt{stop}] = STOP = \{\delta\}$

(iii) $TR[\texttt{a}] = \{\langle a, SKIP\rangle\}$, for each $\texttt{a} \in \texttt{Act}$.

The behavior of TR over complex expressions is given below for each of the functions in the signature. The meaning of $TR[\texttt{p}]$ for a term of the form $f(p_1, p_2, \ldots, p_k)$ is given in terms of its corresponding semantic function f_{TR}. That is: for every f of arity k in $\mathcal{L}$

$$TR[\texttt{f}(\texttt{p}_1, \texttt{p}_2, \ldots, \texttt{p}_k)] = f_{TR}(TR[\texttt{p}_1], TR[\texttt{p}_2], \ldots, TR[\texttt{p}_k])$$

The meaning of the finite functions f_{TR} are now presented. In doing so, we also show that TR satisfies the axioms given above. After we have given the meaning of TR over complex expressions, it can be easily shown that TR is a well-defined function and that it is a homomorphism.

In the interest of readability in the discussion of the properties of the finite functions, we relax our notation slightly in order to permit infix expression of terms such as $f(p_1, p_2, \ldots, p_k)$. Also, we will overload the meaning of the operators + , ; by allowing them to signify either the syntactic construct or the semantic meaning. In general, we will use `p + q` to signify an expression of the language, and $p + q$ to denote its meaning in P_ω.

Sequential Composition In the sequential composition of two expressions, we form a new tree by attaching the tree corresponding to the second argument to all the leaf nodes labelled $SKIP$ of the tree corresponding to first argument. Thus, wherever $SKIP$ appears in the first argument as a leaf node label, this leaf node is replaced by the subtree of the second argument. Formally, the function $;_{TR}$ maps from $P_\omega \times P_\omega$ to P_ω and is defined as follows. For p, q process expressions and $a, b \in Act$

(i) $TR[\texttt{p};\texttt{skip}] = ;_{TR}(TR[\texttt{p}], SKIP) = TR[\texttt{p}]$

(ii) $TR[\texttt{skip};\texttt{p}] = ;_{TR}(SKIP, TR[\texttt{p}]) = TR[\texttt{p}]$

(iii) $TR[\texttt{p};\texttt{stop}] = ;_{TR}(TR[\texttt{p}], STOP) = STOP$

(iv) $TR[\texttt{p};\texttt{q}] = ;_{TR}(TR[\texttt{p}], TR[\texttt{q}]) = \{\langle a, S'\rangle \mid \langle a, S\rangle \in TR[\texttt{p}] \text{ and } S' = S \cdot TR[\texttt{q}])\}$

Where $\cdot$ is set concatenation defined as follows. Let $S_1 = \{t_1, t_2, \cdots\}$ and $S_2 = \{r_1, r_2, \cdots\}$. Then

$$S_1 \cdot S_2 = \{t_i \cdot r_j : t_i \in S_1 \textit{ and } r_j \in S_2\}$$

Without loss of generality, assume $t_i = \langle a, T_i\rangle$ and $r_j = \langle b, R_j\rangle$. Then

$$t_i \cdot r_j = \langle a, \{\langle T_i \cdot b, R_j\rangle\}\rangle$$

Where $\{\langle T_i \cdot b, R_j\rangle\} = STOP$ for each $T_i' \in T_i$ which is equal to $STOP$, and $\{\langle T_i \cdot b, R_j\rangle\} = \langle b, R_j\rangle$ for each $T_i' \in T_i$ which is equal to $SKIP$.

Example

$$\begin{aligned} TR[\texttt{a};\texttt{p}] &= ;_{TR}(\{\langle a, SKIP\rangle\}, TR[\texttt{p}]) \\ &= \{\langle a, S'\rangle \mid \langle a, SKIP\rangle \in \{\langle a, SKIP\rangle\} \\ &\quad \text{and } S' = ;_{TR}(SKIP, TR[\texttt{p}])\} \\ &= \{\langle a, TR[\texttt{p}]\rangle\} \end{aligned}$$

Notice that $TR[\texttt{p};\texttt{skip}] = ;_{TR}(TR[\texttt{p}], SKIP) = TR[\texttt{p}]$ and $TR[\texttt{skip};\texttt{p}] = ;_{TR}(SKIP, TR[\texttt{p}]) = TR[\texttt{p}]$ are special cases of the general case 4 of this definition.

Proposition 1 $;_{TR}$ is well defined.

Proof The proof is by induction of the structure of the process expression p.

Base Case: $p = SKIP$ or $p = STOP$
$SKIP; q = q; SKIP = q \in P_\omega$
$STOP; q = q; STOP = STOP \in P_\omega$
Induction Step:
Assume $;_{TR}$ is well defined for $p' \in P_n$, i.e., $p'; q \in P_\omega$
Let $p = a; p'$
$p; q = \{\langle a, S'\rangle \mid \langle a, S\rangle \in p \text{ and } S' = S; q\}$
Since $p = \{\langle a, p'\rangle\}$,
then $S = p'$ and $S' = p'; q \in P_\omega$ by induction
Therefore, $p; q = \{\langle a, S'\rangle\}$ such that $a \in Act$
and $S' \in P_\omega$
Since $S' \neq \emptyset$ then $S' \in P_\omega - \{\emptyset\}$
Therefore, $p; q$ satisfies Property 1
$\Rightarrow p; q \in P_\omega$ □

Clearly, $;_{TR}$ is not commutative since $TR[\mathtt{a;b}] = \{\langle a, \{\langle b, SKIP\rangle\}\rangle\}$ and this is not equal to $TR[\mathtt{b;a}] = \{\langle b, \{\langle a, SKIP\rangle\}\rangle\}$.

$;_{TR}$ is defined in such a way as to satisfy axioms 1 through 3. $;_{TR}$ can be shown to satisfy axiom 4 (associativity) by a simple induction on r in the equation $p; (q; r) = (p; q); r$.

The inductive step makes use of the fact $\forall p \in P_\omega, (p); a = (p; a)$.

Proposition 2 $;_{TR}$ is associative.

Proof The proof is by induction of the structure of the process expression p.

Base Case: $r = SKIP$ or $r = STOP$
$(p; q); SKIP = p; q = p; (q; SKIP)$
$(p; q); STOP = STOP = p; (q; STOP)$
Induction Step:
Assume $;_{TR}$ is associative for $r' \in P_n$
Let $r = (r'; a)$, then $r \in P_{n+1}$
$(p; q); r = (p; q); r'; a$
$= p; (q; r'); a$ by induction
$= p; (q; r'; a)$
$= p; (q; r)$ □

Non-deterministic Choice Since each process is modeled by a set, the non-deterministic choice between two processes is simply the set-theoretic union of the sets representing the respective argument processes. The function $+_{TR}$ is defined as follows:

for $p, q \in T_\Sigma$ and $a \in Act$

$$TR[\mathtt{p+q}] = +_{TR}(TR[\mathtt{p}], TR[\mathtt{q}]) = TR[\mathtt{p}] \cup TR[\mathtt{q}]$$

where $\cup$ is set union.

Example

$TR[\mathtt{skip+a}] = +_{TR}(TR[\mathtt{skip}], TR[\mathtt{a}]) = SKIP \cup \{\langle a, SKIP\rangle\} = \{SKIP, \langle a, SKIP\rangle\}$

$+_{TR}$ is a well defined function since it corresponds to set-theoretic union in P_ω.

The associativity and commutativity of $+_{TR}$ follow immediately from the corresponding properties of set-theoretic union. Therefore, $+_{TR}$ satisfies axioms 6 and 7. In addition, since $S \cup S = S$, we have idempotency (axiom 5) for $+_{TR}$. That is, $\forall p \in P_\omega,\ p + p = p$.

Properties of the Semantic Function In defining TR over all the elements of the term algebra, we have demonstrated that $\forall \mathtt{p} \in T_\Sigma,\ TR[\mathtt{p}] \in P_\omega$. We are now in a position to show that $\forall \mathtt{p}, \mathtt{q} \in T_\Sigma,\ \mathtt{p} = \mathtt{q} \Rightarrow TR[\mathtt{p}] = TR[\mathtt{q}]$. The proof is by induction on $\mathtt{t} \in T_\Sigma$.

Proposition 3 TR is a function.

Proof

Base Case: $\mathtt{t} = \mathtt{skip}$ or $\mathtt{stop}$
(1) $\mathtt{t_1} = \mathtt{t_2} = \mathtt{skip}$
$TR[\mathtt{t_1}] = SKIP = TR[\mathtt{t_2}]$
(2) $\mathtt{t_1} = \mathtt{t_2} = \mathtt{stop}$
$TR[\mathtt{t_1}] = STOP = TR[\mathtt{t_2}]$
Induction Step:
Assume that TR is a function for $\mathtt{t}$
such that depth($\mathtt{t}$) $\leq n$
(1) Let $\mathtt{t}'_1 = \mathtt{a;t_1}$ and $\mathtt{t}'_2 = \mathtt{a;t_2}$
$TR[\mathtt{t}'_1] = TR[\mathtt{a;t_1}] = \{\langle a, TR[\mathtt{t_1}]\rangle\}$
$TR[\mathtt{t}'_2] = TR[\mathtt{a;t_2}] = \{\langle a, TR[\mathtt{t_2}]\rangle\}$
Then since $\mathtt{t_1} = \mathtt{t_2} \Rightarrow TR[\mathtt{t_1}] = TR[\mathtt{t_1}]$
by induction,
$\mathtt{t}'_1 = \mathtt{t}'_2 \Rightarrow TR[\mathtt{t}'_1] = TR[\mathtt{t}'_2]$
(2) Let $\mathtt{t}''_1 = \mathtt{t_1} + \mathtt{t}'_1$ and $\mathtt{t}''_2 = \mathtt{t_2} + \mathtt{t}'_2$
then $TR[\mathtt{t}''_1] = +_{TR}(TR[\mathtt{t_1}], TR[\mathtt{t}'_1])$
and $TR[\mathtt{t}''_2] = +_{TR}(TR[\mathtt{t_2}], TR[\mathtt{t}'_2])$
and the result follows by induction and the fact that $+_{TR}$ is a function over $P_\omega \times P_\omega$
(3) Let $\mathtt{t}''_1 = \mathtt{t_1;t}'_1$ and $\mathtt{t}''_2 = \mathtt{t_2;t}'_2$
Then the argument for terms of the form $\mathtt{p;q}$ follows from case (1).
Therefore TR is a function. □

The proof that TR is a homomorphism follows immediately from the definition of TR for terms of the form $f(p_1, p_2, \ldots, p_k)$. That is:

for every f of arity k in T_Σ

$$TR[\mathtt{f(p_1, p_2, \ldots, p_k)}] = f_{TR}(TR[\mathtt{p_1}], TR[\mathtt{p_2}], \ldots, TR[\mathtt{p_k}])$$

4.2 The Language $\mathcal{L}_2$: Concurrency

We now extend $\mathcal{L}_1$ to include arbitrary interleaving of the atomic actions of two processes. The new operation is denoted as $\|$. The $\|$ operator is syntactically reduced to the sequential composition and choice operators; that is, we define rewrite rules on terms of the form $\mathtt{p} \;\|\; \mathtt{p}$ such that the resulting term does not contain the parallel operator and its meaning thus defined in terms of composition and choice. Let p and q be processes. Define $\mathtt{p} \;\|\; \mathtt{q}$ by cases:

1. $\mathtt{p} \;\|\; \mathtt{q} \;=\; \mathtt{q} \;\|\; \mathtt{p}$
2. p = skip
   $\mathtt{skip} \;\|\; \mathtt{q} \;\Rightarrow\; \mathtt{q}$
3. p = stop
   $\mathtt{stop} \;\|\; \mathtt{q} \;\Rightarrow\; \mathtt{q;stop}$
4. p = a, q = b
   $\mathtt{a} \;\|\; \mathtt{b} \;\Rightarrow\; \mathtt{(a;b)\ +\ (b;a)}$
5. $\mathtt{p\ =\ (a;p')}$, $\mathtt{q\ =\ (b;q')}$
   $\mathtt{(a;p')} \;\|\; \mathtt{(b;q')} \;\Rightarrow\; \mathtt{(a;(p'} \;\|\; \mathtt{(b;q')))\ +}$
   $\mathtt{(b;((a;p')} \;\|\; \mathtt{q'))}$
6. $\mathtt{q\ =\ (q'\ +\ q'')}$
   $\mathtt{p} \;\|\; \mathtt{(q'\ +\ q'')} \;\Rightarrow\; \mathtt{(p} \;\|\; \mathtt{q')\ +\ (p} \;\|\; \mathtt{q'')}$

4.3 The Language $\mathcal{L}_3$: Synchronization

The intuition of synchronization we wish to formalize is that of the rendezvous mechanism. Because of this, synchronization is only meaningful when a synchronizing process p is put in parallel with another process q with which p may synchronize. σ denotes a request to rendezvous with a parallel process at the entry point $\overline{\sigma}$. $\overline{\sigma} : \mathtt{r}$, signals a readiness to accept a rendezvous and execute critical region r.

We interpret synchronization *per se* as unobservable, i.e. it does not correspond to any element of P_ω. We therefore describe the synchronization operators in terms of rewrite rules on the syntactic elements. Let p, q be processes. Define $\mathtt{p} \;\|\; \mathtt{q}$ by cases:

1. $\mathtt{p} = \mathtt{a};\mathtt{p}'$, $\mathtt{q} = \sigma;\mathtt{q}'$, and $\mathtt{a} \neq \overline{\sigma}$

   $$\mathtt{a};\mathtt{p}' \,\|\, \sigma;\mathtt{q}' \Rightarrow \mathtt{a};(\mathtt{p}' \,\|\, \sigma;\mathtt{q}')$$

   In this case, q has issued a synchronization call to p, but p has not reached the synchronization point. q is held in suspension and only actions from p are executed.

2. $\mathtt{p} = \overline{\sigma};\mathtt{p}'$, $\mathtt{q} = \mathtt{a};\mathtt{q}'$, and $\mathtt{a} \neq \sigma$

   $$\overline{\sigma};\mathtt{p}' \,\|\, \mathtt{a};\mathtt{q}' \Rightarrow \mathtt{a};(\overline{\sigma};\mathtt{p}' \,\|\, \mathtt{q}')$$

   In the complement to the previous case, p is ready to accept a call, but q has not issued one. p is held in suspension and only actions from q are executed. The result is the same when there is a critical region (i.e., $\mathtt{p} = \overline{\sigma} : \mathtt{r};\mathtt{p}'$).

3. $\mathtt{p} = \overline{\sigma};\mathtt{p}'$, $\mathtt{q} = \sigma;\mathtt{q}'$

   $$\overline{\sigma};\mathtt{p}' \,\|\, \sigma;\mathtt{q}' \Rightarrow \mathtt{p}' \,\|\, \mathtt{q}'$$

   Two processes have reached compatible synchronization operators, but there is no critical region. Execution continues as the interleaving of the operations following the synchronization.

4. $\mathtt{p} = \overline{\sigma} : \mathtt{r};\mathtt{p}'$, $\mathtt{q} = \sigma;\mathtt{q}'$

   $$\overline{\sigma} : \mathtt{r};\mathtt{p}' \,\|\, \sigma;\mathtt{q}' \Rightarrow \mathtt{r};(\mathtt{p}' \,\|\, \mathtt{q}')$$

   Two processes have reached compatible synchronization operators, and p contains a critical region r. The critical region is executed; and the continuation is the interleaving of q' with the operations following the critical region.

5. $\mathtt{p} = \mathtt{skip}$, $\mathtt{q} = \sigma;\mathtt{q}'$

   $$\mathtt{skip} \,\|\, \sigma;\mathtt{q}' \Rightarrow \sigma;\mathtt{q}'$$

   If q should be held in suspension while waiting for p to reach a synchronization point, and p terminates without ever reaching such a point, the result is simply $\sigma;\mathtt{q}'$. If this resulting process is in parallel with another process, then there is an opportunity for progress to be made.

6. $\mathtt{p} = \overline{\sigma};\mathtt{p}'$, $\mathtt{q} = \mathtt{skip}$

   $$\overline{\sigma};\mathtt{p}' \,\|\, \mathtt{skip} \Rightarrow \overline{\sigma};\mathtt{p}'$$

   In this case, an accepting process p is held in suspension while waiting for q to reach a synchronization point, and q terminates without ever reaching such a point, resulting in the suspended process p in parallel with skip. The result is $\overline{\sigma};\mathtt{p}'$. If this resulting process is in parallel with another process, then there is an opportunity for progress to be made.

7. $\mathtt{p} = \mathtt{stop}$, $\mathtt{q} = \sigma;\mathtt{q}'$

   $$\mathtt{stop} \,\|\, \sigma;\mathtt{q}' \Rightarrow \sigma;\mathtt{q}';\mathtt{stop}$$

If q should be held in suspension while waiting for p to reach a synchronization point, and p terminates abnormally without ever reaching such a point, the result will be this case: the suspended process q in parallel with stop.

8. $\mathtt{p} = \overline{\sigma}; \mathtt{p}'$, $\mathtt{q} = \mathtt{stop}$

$$\overline{\sigma}; \mathtt{p}' \parallel \mathtt{stop} \Rightarrow \overline{\sigma}; \mathtt{p}'; \mathtt{stop}$$

Finally, there is the case when an accepting process p is held in suspension while waiting for q to reach a synchronization point, and q terminates abnormally without ever reaching such a point, resulting in the suspended process p in parallel with stop. The result is the same when p contains a critical region (i.e., $\mathtt{p} = \overline{\sigma} : \mathtt{r}; \mathtt{p}'$).

5 Infinite Processes

Given the power set structure of P_ω and the Knaster-Tarski fixpoint theorem, we only need to demonstrate that every f_{TR} in the signature is monotonic. Since our domain is algebraic, any function that is monotonic is also continuous. Therefore, the meaning of $\mu X.p$ is obtained as the least upper bound of directed sets in P_ω.

Proposition 4 $+_{TR}$ is monotonic. That is, $(p, q) \sqsubseteq (p', q') \Rightarrow (p + q) \sqsubseteq (p' + q')$.

Proof $p = p'$ and $q \sqsubseteq q'$

(i) $q \subset q'$ then $q \subseteq q' \Rightarrow p \cup q \subseteq p' \cup q' \Rightarrow (p + q) \sqsubseteq' (p' + q')$.

(ii) $q \subset' q'$ then compare $p \cup q$ to $p \cup q'$. This amounts to comparing q and q'. But, $q \sqsubseteq q'$. Therefore, $(p + q) \sqsubseteq (p' + q')$.

Therefore, $+_{TR}$ is monotonic. □

For sequential composition, $;_{TR}$, we first extend the definition given in section 4 to cover the case when the first argument is infinite.

for $p, q \in T_\Sigma$, and p infinite

$$TR[\mathtt{p}; \mathtt{q}] = TR[\mathtt{p}]$$

Proposition 5 (Kleene theorem) The meaning of the process expression $\mu X.f(p)$ is given by

$$TR[\mu X.f(p)] = \bigsqcup_{i \in \omega} f(\emptyset)^i$$

6 Conclusions

We have shown how an algebraic domain of arbitrary trees may be constructed and used to model the behavior of a language supporting non-determinism, parallelism, and synchronization. Thus, we were able to give a formal meaning to finite and infinite processes in a straighforward manner.

We chose to define $\sqsubseteq$ as we did because it captured exactly the relation we wished our processes to have, that is, that of prefix subtrees. It seems natural to relate a process such as $p = a = \{\langle a, SKIP\rangle\}$ with $q = a; (a + b) = \{\langle a, \{\langle a, SKIP\rangle\ \langle b, SKIP\rangle\}\rangle\}$; on the other hand, we wanted to disallow the relation $p' = b = \{\langle b, SKIP\rangle\}$ with $q = a; (a + b)$. Another possible relation would be tree embedding, which in our notation is captured by set inclusion, $\subseteq$. We did, in fact, consider $\subseteq$ when we first developed $\sqsubseteq$; it was rejected because $p = a \not\subseteq q = a; (a + b)$.

References

[1] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambidge University Press, 1990.

[2] J.W. de Bakker, J.A. Bergstra, et al. Linear time and branching time semantics for recursion with merge. In *Automata, Languages and Programming, LNCS 154*, pages 39–51. Springer-Verlag, 1983.

[3] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

[4] Matthew Hennessy. Acceptance trees. *Journal of the ACM*, 32,4:896–928, 1985.

[5] Matthew Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.