Towards Compositional Predicate Transformer Semantics for Concurrent Programs

Eike Best[1]
Institut für Informatik
Universität Hildesheim
D - 3200 Hildesheim

Written on the occasion of the 25th anniversary of Jaco de Bakker's involvement with the Mathematisch Centrum – Centrum voor Wiskunde en Informatica, Amsterdam.

Abstract

For a simple concurrent programming language, a slight modification of the formal framework exposed in [2] is shown to yield a compositional predicate transformer semantics which is consistent with denotational semantics.

1 Introduction

The seminal paper [2] exposes a denotational framework for the definition of the semantics of concurrent programs. We shall explore an idea, due to Jaco de Bakker, to modify this environment to accommodate predicate transformer semantics. For a simple concurrent programming language, we shall show that this modification leads to a generalisation of the well known consistency between relational ('forward') and predicate transformer ('backward') semantics of sequential programs [1].

In section 2 we briefly recall the necessary definitions for sequential programs. Section 3 contains the definitions of forward and backward semantics for simple concurrent programs, along with an example. Section 4 states (and partly proves) the consistency between them. Section 5 contains conclusions.

[1]This work was done while the author was with the Gesellschaft für Mathematik und Datenverarbeitung, D-5205 Sankt Augustin.

2 Sequential programs

Let a be a sequential (possibly nondeterministic) program and let S be the state space of a. The relational semantics $r(a) \subseteq S \times S$ defines $(s, s') \in r(a)$ iff starting with the state s, there is an execution of a terminating in the state s'. The predicate transformer[2] semantics $w(a)\colon 2^S \to 2^S$ defines $w(a, X) = Y$ (for $X, Y \subseteq S$) iff Y is the set of initial states s such that starting with s, every terminating execution of a leads to a final state in X.

Between $r(a)$ and $w(a)$ there is the following relationship [1]:

$$\forall s \in S \forall X \subseteq S\colon s \in w(a, X) \iff r(a, s) \subseteq X. \tag{1}$$

(Informal justification: $s \in w(a, X)$ iff every terminating execution leads into X iff $r(a, s) \subseteq X$.)

For the purpose of explaining our example below, we need to define the relation r and the function w for simple sequential programs of the form $B \to x := e$, that is, assignments guarded by Boolean expressions. We define $(s, s') \in r(B \to x := e)$ iff $B(s) = \mathbf{true}$ and s' equals s^e_x, where s^e_x is the same state as s except that the value of x in s^e_x equals e (as evaluated in s). Moreover, we define $Y = w(B \to x := e, X)$ iff for all $s \in Y$, $B(s) = \mathbf{true}$ implies that $s^e_x \in X$. Unguarded assignments $x := e$ are a special case (consider $\mathbf{true} \to x := e$). Plain Boolean expressions B are special cases as well (consider $B \to \mathbf{skip}$).

3 Simple concurrent programs

Let c denote a shared variable program containing atomic actions as primitives and choice, sequence and parallel composition as combinators. We assume the following syntax for c:

$$c \quad ::= \quad a \mid c_1 \,\square\, c_2 \mid c_1; c_2 \mid c_1 \| c_2,$$

where a denotes an atomic action of the general form $\langle B \to x := e \rangle$. Let S denote the state space of c. For an atomic action a, the relation $r(a) \subseteq S \times S$ and the function $w(a)\colon 2^S \to 2^S$ are defined as in section 2.

As in [2], an object representing the denotation of c will be defined as an element of a domain $\mathcal{P}$ satisfying the following domain equation:

$$\mathcal{P} \quad = \quad \{p_0\} \cup 2_c^{((S \times S) \times \mathcal{P})} \tag{2}$$

where 2_c is the set of all closed subsets (see [2] for the definition of closedness).

[2]Instead of predicates over S we shall consider subsets of S, i.e., elements of 2^S.

Remark:

In [2], the following domain equation has been used instead of equation (2):

$$\mathcal{P}' \quad = \quad \{p_0'\} \cup (S \to 2_c^{(S \times \mathcal{P}')}). \tag{3}$$

However objects $p \in \mathcal{P}$ can be translated equivalently into objects $p' \in \mathcal{P}'$ and vice versa.

To translate p into p', p_0 is translated into p_0' by definition. Let $p = \{(r_1, p_1), \ldots, (r_m, p_m)\} \in \mathcal{P}$ with $p_j \in \mathcal{P}$; then p is translated into $p' = \lambda s.X$ with $(s_i, p_i') \in X$ iff there is a pair $(r_j, p_j) \in p$ such that $(s, s_i) \in r_j$ and p_j is translated into p_i'.

Conversely, to translate p' into p, p_0' is translated into p_0 by definition. Let $p' = \lambda s.\{(s_1, p_1'), \ldots, (s_n, p_n')\} \in \mathcal{P}'$ with $p_i' \in \mathcal{P}'$. Then p' is translated into $p = \{(r_1, p_1), \ldots, (r_n, p_n)\} \in \mathcal{P}$ such that $r_i \subseteq S \times S$ is defined by means of $(s, s') \in r_i \iff (s', p_i) \in p'(s)$, and p_i' is translated into p_i.

These two constructions are inverses of each other [3]. Hence objects in $\mathcal{P}$ and objects in $\mathcal{P}'$ can be used interchangeably. For convenience, we shall use $\mathcal{P}$.

The following will be the defining equation for the semantic domain $\mathcal{Q}$ of the predicate transformer:

$$\mathcal{Q} \quad = \quad \{q_0\} \cup 2_c^{((2^S \to 2^S) \times \mathcal{Q})} \tag{4}$$

Let c be a concurrent program. We define its 'forward' denotation $[\![c\rangle$ and its 'backward' denotation $\langle c]\!]$ as follows:

$$\begin{array}{lcl|rcl}
[\![a\rangle & = & \{(r(a), p_0)\} & (a]\!] & = & \{(w(a), q_0)\} \\
[\![c_1 \,\square\, c_2\rangle & = & [\![c_1\rangle \cup [\![c_2\rangle & \langle c_1 \,\square\, c_2]\!] & = & \langle c_1]\!] \cup \langle c_2]\!] \\
[\![c_1; c_2\rangle & = & [\![c_1\rangle \circ [\![c_2\rangle & \langle c_1; c_2]\!] & = & \langle c_2]\!] \circ \langle c_1]\!] \\
[\![c_1 \| c_2\rangle & = & [\![c_1\rangle \| [\![c_2\rangle & \langle c_1 \| c_2]\!] & = & \langle c_1]\!] \| \langle c_2]\!],
\end{array}$$

where $\|$ is as defined in [2] and $\circ$ is the reverse of the operation defined there (for notational convenience).

As an example, consider the following program c_0:

var x: **integer**;

$$(\underbrace{(x := x + 1)}_{a}; (\underbrace{(x = 0 \to x := 3\rangle}_{b_1} \,\square\, \underbrace{(x \neq 0)}_{b_2})) \,\|\, \underbrace{\langle x := x + 2\rangle}_{d}.$$

Then $[\![c_0\rangle$ is the following set:

$$\begin{array}{rcl} [\![c_0\rangle & = & p_1 = \{(r(a), p_2), (r(d), p_3)\} \\ & & p_2 = \{(r(b_1), p_4), (r(b_2), p_4), (r(d), p_5)\} \\ & & p_3 = \{(r(a), p_5)\} \\ & & p_4 = \{(r(d), p_0)\} \\ & & p_5 = \{(r(b_1), p_0), (r(b_2), p_0)\}. \end{array}$$

On the other hand, $\langle c_0]\!]$ is the following set:

$$\begin{array}{rcl} \langle c_0]\!] & = & q_1 = \{(w(b_1), q_2), (w(b_2), q_2), (w(d), q_3)\} \\ & & q_2 = \{(w(a), q_4), (w(d), q_5)\} \\ & & q_3 = \{(w(b_1), q_5), (w(b_2), q_5)\} \\ & & q_4 = \{(w(d), q_0)\} \\ & & q_5 = \{(w(a), q_0)\}. \end{array}$$

Both expressions can be viewed as labelled trees [2]; however, despite the close symmetry in the definition, these trees are not, in general, isomorphic. The objects $[\![c\rangle$ and $\langle c]\!]$ contain enough information for the relational semantics and the predicate transformer semantics of c to be easily derivable. To this end we define two semantic functions ρ and ϕ with the following functionality:

$$\begin{array}{ll} \rho: & \mathcal{P} \times S \rightarrow 2^S \\ \phi: & \mathcal{Q} \times 2^S \rightarrow 2^S. \end{array}$$

Let $p \in \mathcal{P}$ and $s \in S$. Then $\rho(p, s)$ is defined recursively as follows:

$$\begin{array}{rcl} \rho(p_0, s) & = & \{s\} \\ \rho(p, s) & = & \bigcup\{\rho(p', t) \mid \exists r \subseteq S \times S : (r, p') \in p \wedge (s, t) \in r\}. \end{array} \tag{5}$$

Let $q \in \mathcal{Q}$ and $X \subseteq S$. Then $\phi(q, X)$ is defined recursively as follows:

$$\begin{array}{rcl} \phi(q_0, X) & = & X \\ \phi(q, X) & = & \bigcap\{\phi(q', Y) \mid \exists w : 2^S \rightarrow 2^S : (w, q') \in q \wedge Y = w(X)\}. \end{array} \tag{6}$$

The intention is that $\rho(p, s)$ describes the set of final states that are reachable from s by execution paths through p. For a final state to be included in this set, it suffices that there is one path by which it is reachable; hence the union quantifier in formula (5). On the other hand, $\phi(q, X)$ describes the intersection of all sets of initial states that are reachable by paths through q. The intention is that this set describes the predicate transformer of X backward through p (where p is related to q as $[\![c\rangle$ is related to $\langle c]\!]$). For an initial state to be in it, all possible execution paths through p have to lead into X; hence the intersection quantifier in formula (6).

As an example, reconsider the program c_0 with $p_1 = [c\rangle$ and $q_1 = \langle c]$. Let s denote the initial state $x = -1$. For convenience, we will denote states $x = v$ simply by v (hence s denotes the state -1). We compute $\rho(p_1, -1)$:

$$\begin{array}{lcl} \rho(p_1,-1) & = & \bigcup\{\rho(p_2,0), \rho(p_3,1)\} \\ & = & \rho(p_2,0) \cup \rho(p_3,1) \\ \rho(p_2,0) & = & \rho(p_4,3) \cup \rho(p_5,2) \\ \rho(p_3,1) & = & \rho(p_5,2) \\ \rho(p_4,3) & = & \rho(p_0,5) = \{5\} \\ \rho(p_5,2) & = & \rho(p_0,2) = \{2\}. \end{array}$$

Hence $\rho(p_1, -1) = \{2, 5\}$.
Let X denote the set of final states $\{2,3,4,5\}$. We compute $\phi(q_1, X)$:

$$\begin{array}{lcl} \phi(q_1,\{2,3,4,5\}) & = & \phi(q_2,S) \cap \phi(q_2,\{0,2,3,4,5\}) \cap \phi(q_3,\{0,1,2,3\}) \\ & = & \phi(q_2,\{0,2,3,4,5\}) \cap \phi(q_3,\{0,1,2,3\}) \\ \phi(q_2,\{0,2,3,4,5\}) & = & \phi(q_4,\{-1,1,2,3,4\}) \cap \phi(q_5,\{-2,0,1,2,3\}) \\ \phi(q_3,\{0,1,2,3\}) & = & \phi(q_5,S) \cap \phi(q_5,\{0,1,2,3\}) \\ & = & \phi(q_5,\{0,1,2,3\}) \\ \phi(q_4,\{-1,1,2,3,4\}) & = & \phi(q_0,\{-3,-1,0,1,2\}) = \{-3,-1,0,1,2\} \\ \phi(q_5,\{0,1,2,3\}) & = & \phi(q_0,\{-1,0,1,2\}) = \{-1,0,1,2\}. \end{array}$$

Hence $\phi(q_1, \{2,3,4,5\}) = \{-1,0,1,2\}$, as could be expected.

4 Consistency

In this section we state (and partially prove) a generalisation of equation (1). Let c be a concurrent program and let S be its state space as in the previous section. Then we claim that the following holds:

$$\forall s \in S\ \forall X \subseteq S\colon \underbrace{s \in \phi(\langle c], X)}_{lhs} \iff \underbrace{\rho([\![c\rangle, s) \subseteq X}_{rhs}. \qquad (7)$$

In the proof of (7), we may proceed by structural induction over the syntax of c. However, we deal only with the two cases $c = a$ and $c = c_1 \,\square\, c_2$.

Case 1: $c = a$.

$$\begin{array}{lcll} lhs & = & s \in \phi(\langle c], X) & \\ & \iff & s \in \phi(\{(w(a), q_0)\}, X) & \text{(Def. } \langle x]) \\ & \iff & s \in \phi(q_0, w(a, X)) & \text{(Def. } \phi) \\ & \iff & s \in w(a, X) & \text{(Def. } \phi) \\ rhs & = & \rho([\![c\rangle, s) \subseteq X & \\ & \iff & \rho(\{(r(a), p_0)\}, s) \subseteq X & \text{(Def. } [\![c\rangle) \\ & \iff & (\bigcup\{\rho(p_0, t) \mid (s,t) \in r(a)\}) \subseteq X & \text{(Def. } \rho) \\ & \iff & (\bigcup\{\{t\} \mid (s,t) \in r(a)\}) \subseteq X & \text{(Def. } \rho) \\ & \iff & r(a, s) \subseteq X & \text{(rewriting).} \end{array}$$

Now the claim that *lhs* $\iff$ *rhs* follows directly from equality (1).

Case 2: $c = c_1 \,[]\, c_2$.

The proof is conducted in two steps. First, *lhs* and *rhs* are rewritten in a more convenient form amenable to induction. Then, their equality is shown using the induction hypothesis for c_1 and c_2.

$$
\begin{array}{lcll}
\mathit{lhs} & = & s \in \phi(\langle c \rceil, X) & \\
& \iff & s \in \phi(\langle c_1 \rceil \cup \langle c_2 \rceil, X) & \text{(Def. } \langle c \rceil\text{)} \\
& \iff & s \in \bigcap\{\phi(q', Y) \mid \exists w{:}\, 2^S \to 2^S{:} & \\
& & \quad (w, q') \in (\langle c_1 \rceil \cup \langle c_2 \rceil) \wedge Y = w(X)\} & \text{(Def. } \phi\text{)} \\
& \iff & \forall q' \in \mathcal{Q}\, \forall Y \subseteq S{:}\, [\exists w{:}\, 2^S \to 2^S{:} & \\
& & \quad (w, q') \in \langle c_i \rceil \wedge Y = w(X)] \Rightarrow s \in \phi(q', Y) & \text{(for } i = 1, 2\text{)}
\end{array}
$$

$$
\begin{array}{lcll}
\mathit{rhs} & = & \rho(\lfloor c \rangle, s) \subseteq X & \\
& \iff & X \supseteq \rho(\lfloor c_1 \rangle \cup \lfloor c_2 \rangle, s) & \text{(Def. } \lfloor c \rangle\text{)} \\
& \iff & X \supseteq (\bigcup\{\rho(p', t) \mid \exists r \subseteq S \times S{:} & \\
& & \quad (r, p') \in (\lfloor c_1 \rangle \cup \lfloor c_2 \rangle) \wedge (s, t) \in r\}) & \text{(Def. } \rho\text{)} \\
& \iff & \forall p' \in \mathcal{P}\, \forall t \in S{:}\, [\exists r \subseteq S \times S{:} & \\
& & \quad (r, p') \in \lfloor c_i \rangle \wedge (s, t) \in r] \Rightarrow \rho(p', t) \subseteq X & \text{(for } i = 1, 2\text{)}
\end{array}
$$

Next we prove *lhs* $\Rightarrow$ *rhs*.

Consider any $q' \in \mathcal{Q}$ and $w{:}\, 2^S \to 2^S$ such that $(w, q') \in \langle c_i \rceil$, and define $Y = w(X)$. Then by *lhs*, we have $s \in \phi(q', Y)$. Hence s is in the intersection of all sets $\phi(q', Y)$ such that q' and Y have the above properties and hence, by the definition of ϕ, we have $s \in \phi(\langle c_i \rceil, X)$. By induction hypothesis, $\rho(\lfloor c_i \rangle, s) \subseteq X$ holds true. Hence by the definition of ρ, we have $\rho(p', t) \subseteq X$ whenever $p' \in \mathcal{P}$, $t \in S$ and $r \subseteq S \times S$ are such that $(r, p') \in \lfloor c_i \rangle$ and $(s, t) \in r$. But this means that *rhs* holds.

The proof of *rhs* $\Rightarrow$ *lhs* is similar.

5 Conclusions

When $\lfloor c \rangle$ and $\langle c \rceil$ are viewed as labelled trees then their relationship can be characterised by 'tree inversion', an inverse tree to a given tree being determined by having the reverse edge walks, with appropriately matched edge labels. Although tree inversion in this sense is not unique, there exists an inductive definition for it [3]. By means of this inductive definition, it becomes possible to derive relational semantics and predicate transformer semantics from each other, rather than from the common underlying program.

It appears difficult to uphold the idea of tree inversion for iterative or recursive concurrent programs, because trees containing infinite paths cannot

be inverted. Hence this author's preferred approach is to suggest – syntactic or semantic – means of guaranteeing the termination of any iterative or recursive loops. This would lead to trees which may be infinitely broad but are always finitely long and can be inverted, quite in contrast to the case of sequential boundedly nondeterministic programs which lead to finitely broad (but possibly infinitely long) trees.

Acknowledgement

The work described in this short note was directly stimulated by Jaco de Bakker's questions and remarks. It was done during an enjoyable stay of the author in Amsterdam in October 1983. At that time we were not aware of any other work on predicate transformers for concurrent programs. In the meantime, and independently, at least two other papers have been published on the subject: [4] and [5].

References

[1] J.W. de Bakker: *Mathematical Theory of Program Correctness.* Prentice Hall (1980).

[2] J.W. de Bakker und J. Zucker: *Processes and the Denotational Semantics of Concurrency.* Information and Control, Vol.54, No.1/2, pp.70-120 (1982).

[3] E. Best: *An Equivalence Between Plotkin's Term Rewrite Semantics, Control Sequence Semantics, and the Process Semantics of de Bakker and Zucker.* BEGRUND-Memorandum No.33 (April 1984). *Towards a General Equivalence of 'Forward' Semantics and Predicate Transformer Semantics for Concurrent Programs.* BEGRUND-Memorandum No.34 (April 1984).

[4] T. Elrad and N. Francez: *A Weakest Precondition Semantics for Communicating Processes.* TCS 29, pp.231-250 (1984).

[5] L. Lamport: win *and* sin*: Predicate Transformers for Concurrency.* Research Report No.17, Digital Equipment Corporation, Systems Research Center (May 1987). (To appear in TOPLAS.)