A Note on Disjoint Parallelism

Krzysztof R. Apt
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam
The Netherlands
and
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712–1188
U.S.A.

Ernst-Rüdiger Olderog
Department of Computer Science
Christian-Albrechts University
2300 Kiel 1, Olshausenstrasse 40
West Germany

Abstract: We prove determinism of disjoint parallel programs by applying simple results on abstract reduction systems.

In this note we consider disjoint parallel programs as defined in Hoare [1972]. We assume from the reader the knowledge of **while**-programs (see e.g. De Bakker [1980]) and their semantics defined by means of transitions (see Hennessy and Plotkin [1979]). Two **while**-programs S_1 and S_2 are called *disjoint* if none of them can change the variables accessed by the other one, i.e. if

$$change(S_1) \cap var(S_2) = var(S_1) \cap change(S_2) = \emptyset,$$

where $change(S)$ is the set of variables of S which can be modified by it, i.e. to which a value is assigned within S by means of an assignment. Note that disjoint programs are allowed to read the same variables. For example, the programs

$$x := z \text{ and } y := z$$

are disjoint because $change(x := z) = \{x\}$, $var(y := z) = \{y, z\}$ and $var(x := z) = \{x, z\}, change(y := z) = \{y\}$.

On the other hand the programs $x := z$ and $y := x$ are not disjoint because $x \in change(x := z) \cap var(y := x)$.

Disjoint parallel programs are generated by the same clauses as those defining **while**-programs together with the following clause for *parallel composition*:

$$S ::= [S_1 \| \ldots \| S_n]$$

where for $n \geq 1$, $S_1, \ldots, S_n$ are pairwise disjoint **while**-programs. Thus we do not allow nested parallelism, but we allow parallelism to occur within sequential composition, conditional statements and **while**-loops.

Intuitively, a disjoint parallel program of the form $S \equiv [S_1 \| \ldots \| S_n]$ terminates if and only if all of its components $S_1, \ldots, S_n$ terminate; the final state is then the composition of the final states of $S_1, \ldots, S_n$.

Following Hennessy and Plotkin [1979] we define the semantics of disjoint parallel programs in terms of transitions. Intuitively, a disjoint parallel program $[S_1 \| \ldots \| S_n]$ performs a transition if one of its component performs a transition. Formally, we expand the transition system for **while**-programs by the following rule:

$$\frac{< S_i, \sigma > \; \rightarrow \; < T_i, \tau >}{< [S_1 \| \ldots \| S_i \| \ldots \| S_n], \sigma > \; \rightarrow \; < [S_1 \| \ldots \| T_i \| \ldots \| S_n], \tau >}$$

where $1 \leq i \leq n$.

Recall that computations of disjoint parallel programs are defined as those of **while**-programs. For example

$$\begin{array}{ll} & < [x := 1 \| y := 2 \| z := 3], \sigma > \\ \rightarrow & < [E \| y := 2 \| z := 3], \sigma[1/x] > \\ \rightarrow & < [E \| E \| z := 3], \sigma[1/x][2/y] > \\ \rightarrow & < [E \| E \| E], \sigma[1/x][2/y][3/z] > \end{array}$$

is a computation of $[x := 1 \| y := 2 \| z := 3]$ starting in σ.

Here E stands for the empty program and its occurrence denotes termination of the appropriate component. For example, $[E \| y := 2 \| z := 3]$ denotes a parallel program where the first component has terminated. As explained above a parallel program terminates if and only if all its components terminate. Consequently we identify

$$[E \| \ldots \| E] \equiv E.$$

Thus the final configuration in the above computation is the terminating configuration

$$< E, \sigma[1/x][2/y][3/z] > .$$

We define the partial and total correctness semantics $\mathcal{M}$ and $\mathcal{M}_{tot}$ of disjoint parallel programs by putting for a state σ

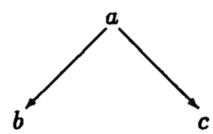
$$\mathcal{M}[[S]](\sigma) = \{\tau \mid < S, \sigma > \rightarrow^* < E, \tau >\}$$

where $\rightarrow^*$ denotes the transitive reflexive closure of $\rightarrow$, and

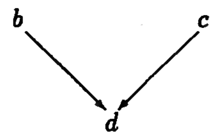
$$\begin{aligned} \mathcal{M}_{tot}[[S]](\sigma) \quad = \quad & \mathcal{M}[[S]](\sigma) \\ \cup \quad & \{\bot \mid S \text{ can diverge from } \sigma\}. \end{aligned}$$

Our aim is to prove that for a disjoint parallel program only one outcome for a given initial state is possible. In other words, for any disjoint parallel program S and state σ, $\mathcal{M}_{tot}[[S]](\sigma)$ has exactly one element. To this end, we need some results concerning abstract reduction systems.

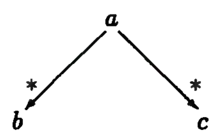
Definition 1 Let $\rightarrow$ be a non-empty binary relation. Denote by $\rightarrow^*$ the transitive reflexive closure of $\rightarrow$. $\rightarrow$ satisfies the *diamond property* if for all a, b, c such that $b \neq c$



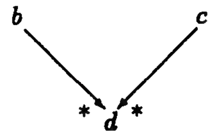
implies that for some d



$\rightarrow$ is called *confluent* if for all a, b, c



implies that for some d



□

The following lemma due to Newman [1942] is of importance to us.

Lemma 2 (Newman's Lemma) If a relation $\rightarrow$ satisfies the diamond property then it is confluent.

Proof. Suppose that $\rightarrow$ satisfies the diamond property. Let $\rightarrow^n$ stand for the n-fold composition of $\rightarrow$. A straightforward proof by induction on $n \geq 0$ shows that $a \rightarrow b$ and $a \rightarrow^n c$ implies that for some $i \leq n$ and some d, $b \rightarrow^i d$ and $c \rightarrow^\epsilon d$. Here $c \rightarrow^\epsilon d$ iff $c \rightarrow d$ or $c = d$. Thus $a \rightarrow b$ and $a \rightarrow^* c$ implies that for some d, $b \rightarrow^* d$ and $c \rightarrow^* d$.

This implies by induction on $n \geq 0$ that if $a \rightarrow^* b$ and $a \rightarrow^n c$ then for some d, $b \rightarrow^* d$ and $c \rightarrow^* d$. This proves confluence. □

We shall also need the following lemma.

Lemma 3 Suppose $\rightarrow$ satisfies the diamond property and that $a \rightarrow b$, $a \rightarrow c$, $b \neq c$. If there exists an infinite sequence $a \rightarrow b \rightarrow \ldots$, then there exists an infinite sequence $a \rightarrow c \rightarrow \ldots$.

Proof. Consider an infinite sequence $a_0 \rightarrow a_1 \rightarrow \ldots$ where $a_0 = a$ and $a_1 = b$.
Case 1. For some $i \geq 0$, $c \rightarrow^* a_i$.

Then $a \rightarrow c \rightarrow^* a_i \rightarrow \ldots$ is the desired sequence.

Case 2. For no $i \geq 0$, $c \rightarrow^* a_i$.

By induction on i we construct an infinite sequence $c_0 \rightarrow c_1 \rightarrow \ldots$ such that $c_0 = c$ and for all $i \geq 0$, $a_i \rightarrow c_i$. For $i = 0$, c_i is already correctly defined.

Consider now the induction step. We have $a_i \rightarrow a_{i+1}$ and $a_i \rightarrow c_i$ for some $i > 0$. Also, since $c \rightarrow^* c_i$, by the assumption $c_i \neq a_{i+1}$. Again by the diamond property for some c_{i+1}, $a_{i+1} \rightarrow c_{i+1}$ and $c_i \rightarrow c_{i+1}$. □

Define now for an element a in the domain of $\rightarrow$

$$\begin{array}{rcl} \mathit{yield}(a) & = & \{b \mid a \rightarrow^* b, b \text{ is } \rightarrow\text{-maximal}\} \\ & \cup & \{\bot \mid \text{ there exists an infinite sequence } a \rightarrow a_1 \rightarrow \ldots\} \end{array}$$

where b is called $\rightarrow$-maximal if for no c, $b \rightarrow c$.

Lemma 4 Suppose that $\rightarrow$ satisfies the diamond property. Then for every a, $yield(a)$ has exactly one element.

Proof. Suppose that for some $\rightarrow$-maximal b and c, $a \rightarrow^* b$ and $a \rightarrow^* c$. By Newman's Lemma for some d, $b \rightarrow^* d$ and $c \rightarrow^* d$. By the $\rightarrow$-maximality of b and c, both $b = d$ and $c = d$, i.e. $b = c$.

Thus the set $\{b \mid a \rightarrow^* b, b \text{ is } \rightarrow\text{-maximal}\}$ has at most one element. If it is empty, then $yield(a) = \{\perp\}$ and we are done.

Otherwise it has exactly one element, say b. Assume by contradiction that there exists an infinite sequence $a \rightarrow a_1 \rightarrow \ldots$. Consider a sequence $b_0 \rightarrow b_1 \rightarrow \ldots \rightarrow b_k$ where $b_0 = a$ and $b_k = b$. Then $k > 0$. Let $b_0 \rightarrow \ldots \rightarrow b_\ell$ be the longest prefix of $b_0 \rightarrow \ldots \rightarrow b_k$ which is an initial fragment of an infinite sequence $a \rightarrow c_1 \rightarrow \ldots$. Then ℓ is well defined, $b_\ell = c_\ell$ and $\ell < k$, since b_k is $\rightarrow$-maximal. Thus $b_\ell \rightarrow b_{\ell+1}$ and $b_\ell \rightarrow c_{\ell+1}$. By the definition of ℓ, $b_{\ell+1} \neq c_{\ell+1}$. By Lemma 3 there exists an infinite sequence $b_\ell \rightarrow b_{\ell+1} \rightarrow \ldots$. This contradicts the choice of ℓ. Thus $yield(a) = \{b\}$. □

We now wish to apply Lemma 4 to the case of disjoint parallel programs. To this purpose we prove first the following lemma.

Lemma 5 (Diamond Property). Let S be a disjoint parallel program and σ a state. Whenever

$$\begin{array}{c} < S, \sigma > \\ \swarrow \quad \searrow \\ < S_1, \sigma_1 > \neq < S_2, \sigma_2 >, \end{array}$$

then for some configuration $< T, \tau >$

$$\begin{array}{c} < S_1, \sigma_1 > \quad < S_2, \sigma_2 > \\ \searrow \quad \swarrow \\ < T, \tau >. \end{array}$$

Proof. By the format of the transition rules, S is of the form $[T_1 \| \ldots \| T_n]$ for some pairwise disjoint **while**-programs $T_1, \ldots, T_n$ and there exist **while**-programs T_i' and T_j', with $i \neq j$, $1 \leq i, j \leq n$ such that

$$\begin{array}{l} S_1 = [T_1 \| \ldots \| T_i' \| \ldots \| T_n], \\ S_2 = [T_1 \| \ldots \| T_j' \| \ldots \| T_n], \\ < T_i, \sigma > \; \rightarrow \; < T_i', \sigma_1 >, \\ < T_j, \sigma > \; \rightarrow \; < T_j', \sigma_2 > . \end{array}$$

Let

$$T = [T_1' \| \ldots \| T_n']$$

where for $k = 1, \ldots, n$ such that $k \neq i$ and $k \neq j$

$$T'_k = T_k$$

If $\sigma_1 \neq \sigma$ then the transition $< S, \sigma > \ \rightarrow \ < S_1, \sigma_1 >$ consists of executing an assignment statement and then $\sigma_1 = \sigma[d_1/u_1]$ for some variable u_1 and element d_1 of the domain D.

Similarly if $\sigma_2 \neq \sigma$ then $\sigma_2 = \sigma[d_2/u_2]$ for some variable u_2 and element d_2 of the underlying domain over which all variables range. Here $\sigma[d/u]$ stand for the state differing from σ only on the variable u to which it assigns the value d.

We now define τ depending on the cardinality of the set $\{\sigma, \sigma_1, \sigma_2\}$:

$$\tau = \begin{cases} \sigma & \text{if } \ card\{\sigma, \sigma_1, \sigma_2\} = 1, \\ \rho & \text{if } \ card\{\sigma, \sigma_1, \sigma_2\} = 2 \text{ and} \\ & \quad \{\sigma, \sigma_1, \sigma_2\} = \{\sigma, \rho\}, \\ \sigma[d_1/u_1][d_2/u_2] & \text{if } \ card\{\sigma, \sigma_1, \sigma_2\} = 3. \end{cases}$$

If $\tau = \sigma[d_1/u_1][d_2/u_2]$ then $< S_1, \sigma_1 > \ \rightarrow \ < T, \tau >$. But by the disjointness condition $\tau = \sigma[d_2/u_2][d_1/u_1]$ which proves that also $< S_2, \sigma_2 > \ \rightarrow \ < T, \tau >$.

Other cases are straightward and left to Jaco. □

As an immediate corollary we obtain the desired result.

Theorem 6 (Determinism) For every disjoint parallel program and a state σ, $\mathcal{M}_{tot}[[S]](\sigma)$ has exactly one element.

Proof. By Lemma's 4 and 5. □

References

[1] J.W. de Bakker [1980], Mathematical Theory of Program Correctness, Prentice-Hall, Englewood Cliffs, N.J., 1980.

[2] M.C.B. Hennessy and G.D. Plotkin [1979], Full abstraction for a simple programming language, in: Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 74 (J.Bečvar, ed.), pp.108-120, 1979.

[3] C. A. R. Hoare [1972], Towards a theory of parallel programming, in: Operating Systems Techniques (C.A.R. Hoare, R.H. Perrot, eds.), pp. 61–71, Academic Press, 1972.

[4] M.H.A. Newman [1942], On theories with a combinatorial definition of "equivalence", *Ann. Math, 43*, pp. 223–243, 1942.