Automating the Refinement of Specifications for Distributed Systems via Syntactic Transformations [1]

Paul Attie, Champak Das

Florida International University
School of Computer Science
Miami, Florida 33199
{attie,cdas01}@fiu.edu

Abstract

The idea of successively refining an abstract specification until it contains enough detail to suggest an implementation has been investigated by numerous researchers. The emphasis to date has been on techniques that, unfortunately, lead to a large amount of manual formal labor for each refinement step. With such techniques, both the cost and the possibility of errors arising in formal manipulation are high. Using a theorem prover can reduce the number of manipulation errors, but, given current technology, the amount of labor is still daunting. This research explores an alternative solution to the refinement problem, namely the use of syntactic transformations to realize each refinement step. We reduce formal labor by employing automatic transformations that guarantee the preservation of desirable properties — e.g., deadlock-freedom. Automatic transformations are particularly appealing for the development of large, complex distributed systems, where a manual approach to refinement would be prohibitively expensive. Distributed computations are, by nature, reactive and concurrent, so their correctness cannot be specified as a simple functional relationship between inputs and outputs. Instead, specifications must describe the time-varying behavior of the system. Further difficulty is caused by the fact that such important characteristics of distributed systems as deadlock-freedom are global properties that cannot be achieved through considering local structures only. Transformations generally must encompass the entire system. This paper presents two syntactic transformations — the left-sequence introduction and the right-sequence introduction — and demonstrates that they preserve deadlock-freedom.

Keywords: Concurrent Program, Deadlock-Freedom, Design Transformation, Multiparty Interaction, Refinement.

1 Introduction

Formal program verification is widely accepted as a means of guaranteeing the correctness of concurrent programs [Hoare 69, Francez 92, Lamport 80, Vardi 87]. The practical utility of formal verification is limited by numerous factors — for example, the large amount of manual labor required, the possibility of proof errors, the lack of personnel trained in proof techniques, and so on. It is also clear that post-development verification alone does not provide a systematic software development process. Successive refinement is an alternative approach for producing correct concurrent programs: start with an abstract specification and incrementally refine it to a stage where implementation becomes relatively straightforward. Refinement is not a new idea, of course, but most of the techniques proposed to date (for example, see [Back et. al. 83, Back et. al. 85, Chandy et. al. 88, Ramesh et. al. 87]) require large amounts of manual formal labor for each refinement step. Even methodologies based on automatic theorem proving [Manna et. al. 94, Constable et. al. 89, Cleaveland et. al. 96] require user intervention, either to select the rule of inference used to generate the next step of a proof [Cleaveland et. al. 96],

[1] Both authors were supported in part by Rome Laboratory, U. S. Air Force, under Contract No. F30602-93-C-0247.

or to supply invariants and/or correct automatically generated invariants that are not "inductive," i.e., that cannot be proven to be invariants in the deductive system being used [Manna et. al. 94]. Other approaches [Aceto 92, van Glabeek 90, Czaja et. al. 91] address the issue of which equivalence relations are preserved by refinement. In other words, if P and Q are programs such that P is "bisimilar" to Q (under some notion of bisimulation, see [Baeten et. al. 90, Milner 89]), then $ref(P)$ will be bisimilar to $ref(Q)$ under this same bisimulation notion, where $ref(P)$ and $ref(Q)$ are "corresponding" refinements of P, Q, i.e., refinements that refine the same action of P, Q in the same way. While such approaches provide a nice theoretical foundation, they do not directly address the central concern, namely the establishment of a relationship between a program and its refinement, i.e., between P and $ref(P)$.

Central to our approach is the concept of *correctness-preserving syntactic transformations.* Such transformations are mechanizable and, therefore, do not involve significant amounts of manual labor. Using this approach, the process of development may be viewed as the human-assisted high-level compilation of a specification into code. Furthermore, by avoiding proof-based methods, we obviate the need to formulate (usually) complicated invariants, a difficult task at best, even with the aid of automated tools.

In the foreseeable future, human creativity will remain essential for choosing an appropriate transformation to apply at each stage. But verifying that a transformation preserves desired properties is unnecessary, in our approach, because this is guaranteed by the fact that the transformations are correctness-preserving.

This paper presents transformations that decompose an action into a sequence of actions. The correctness property that these transformations are guaranteed to preserve is freedom from deadlock. Formal proofs of this characteristic are given, as well as an example of refinement using the transformations.

2 Notation, Syntax, and Operational Semantics

A *program* is the composition of a fixed set of sequential processes executing concurrently. We use the nondeterministic interleaving model of concurrency. That is, we view concurrency as the nondeterministic interleaving of events. An *event* is the atomic (i.e., indivisible) execution of an action, described in Definition 1. We use $;$, $[\!]$, $\|$ to denote sequence, choice, and parallel composition, respectively. The semantics of these operators is similar to that given in CSP [Hoare 85]. To model state transitions, we employ the concept of a labeled transition system, as used in [Milner 89]. $\xrightarrow{a}$ will denote the transition relation induced by action a. The formal meaning of $;$, $[\!]$, $\|$, $\xrightarrow{a}$ is given below.

Definition 1 (*Action*) *An* action*, a consists of a character string, (i.e., an identifier) drawn from some set, $\mathcal{A}$, of identifiers.*

We use lower-case letters towards the beginning of the alphabet to denote actions.

Definition 2 (*Action Expression*) *An* action expression *E is a finite expression given by the following BNF grammar:*

> $<$*action_expression*$>$ ::=
>
> $<$*action_expression*$>$ $[\!]$ $<$*action_expression*$>$ |
>
> $<$*action_expression*$>$; $<$*action_expression*$>$ |
>
> ($<$*action_expression*$>$) |
>
> $<$*action*$>$ | ε | 0

E, F, G, H range over the set of action expressions. We make the convention that ; has higher binding power than $[\!]$, so that $E; F \; [\!] \; G$ denotes $(E; F) \; [\!] \; G$. Intuitively, $E; F$ means execute E and then execute F, while $E \; [\!] \; F$ means execute either E or F. $[\!]$ is commutative, and $[\!]$, ; are both associative.

0, ("Stop"), is the identity element of $[\!]$, and ε ("Skip"), is the identity element of ;. They obey the following axioms: $0 \; [\!] \; A = A$, $A \; [\!] \; 0 = A$, $A; \varepsilon = A$, $\varepsilon; A = A$. We define the relation of equality ($=$) among action expressions as follows. $E = F$ iff one can be obtained from the other by a finite number of any of the following: 1) application of the above axioms for 0 and ε, 2) application of the commutativity property of $[\!]$ or the associativity property of $[\!]$, ;, and 3) adding/removing parentheses in accordance with the precedence of ; over $[\!]$.

We define αE, the *alphabet* of action expression E, as follows.

Definition 3 (*Alphabet*) *The* alphabet *of an action expression is given as follows*:

$$alphabet(a) \overset{\text{df}}{=} \{a\}$$
$$alphabet(E \; [\!] \; F) \overset{\text{df}}{=} alphabet(E) \cup alphabet(F)$$
$$alphabet(E; F) \overset{\text{df}}{=} alphabet(E) \cup alphabet(F)$$

Definition 4 (*Sequential Process*) *A* sequential process*, P_i, consists of a process body and a process alphabet. The process body, $body(P_i)$, is an expression of the form $F_i; *E_i$ where F_i, E_i are action expressions. The process alphabet, $alphabet(P_i)$, is defined to be a set of action names. The process alphabet must contain $alphabet(F_i) \cup alphabet(E_i)$.*

Note that this definition extends the definition of "alphabet" to processes. We have also introduced "*", which denotes infinite iteration. We extend $=$ to process bodies in a straightforward manner. If $body(P_i) = F_i; *E_i$, and $body(P_j) = F_j; *E_j$, then $body(P_i) = body(P_j)$ iff $F_i = F_j$ and $E_i = E_j$. Finally, $P_i = P_j$ iff $alphabet(P_i) = alphabet(P_j)$ and $body(P_i) = body(P_j)$. (Note that when we write $alphabet(P_i) = alphabet(P_j)$, the $=$ symbol denotes standard set-theoretic equality, because alphabets are sets.)

Definition 5 (*Program*) *A* program*, P, is the parallel composition of one or more sequential processes; i.e., $P = (\| \; i \in \varphi : P_i)$, where φ is some suitable index set. Also, $alphabet(P) = (\cup i \in \varphi : alphabet(P_i))$.*

$\|$ is commutative and associative, which justifies the index notation $\| \; i \in \varphi$ introduced in the above definition. For sake of simplicity, we assume that all variables in a program are uniquely named. We extend $=$ to programs in the expected manner: $(\| \; i \in \varphi : P_i) = (\| \; i \in \psi : Q_i)$ iff $\varphi = \psi$ and, for all $i \in \varphi, P_i = Q_i$.

Definition 6 (*Participant Set PA_P*) *The* participant set $PA_P(a)$ *of action a is given by*:

$$PA_P(a) \overset{\text{df}}{=} \{i \mid a \in \mathit{alphabet}(P_i)\}$$

$PA_P(a)$ is the set of processes within program P that jointly and synchronously participate in the execution of action a. If $|PA_P(a)| > 1$ then a is a *multiparty interaction* of program P. If $|PA_P(a)| = 1$, then a is a *local action* of some process P_i (namely the P_i such that $a \in \mathit{alphabet}(P_i)$) in program P.

2.1 Operational Semantics

The operational semantics of a program P is defined by giving the transitions that the execution of each action a in $\mathit{alphabet}(P)$ can generate. Our definition proceeds bottom up, defining the binary transition relation $\xrightarrow{a}$ over action expressions first, then over sequential processes, and finally over programs. In each case, execution of a takes the action expression (sequential process, program) to a new action expression (sequential process, program resp.). In order to avoid the well-known phenomenon that the behavior of $E \; [\!] \; F$ and $\varepsilon; E \; [\!] \; \varepsilon; F$ is different even though they are "equal", we stipulate that the transition relation cannot be applied to 0 and ε, i.e., $\xrightarrow{\varepsilon}$ and $\xrightarrow{0}$ are not defined. This does not cause any difficulties, since ε and 0 can always be eliminated from an expression using the above axioms, after which the transition relation can be applied. This stipulation means that 0 and ε are never executed.

Definition 7 (*Transition Relation $\xrightarrow{a}$*) *The transitions generated by action a are as follows*:

Act. $$\frac{}{(a; E) \xrightarrow{a} E}$$

Ch. $$\frac{E \xrightarrow{a} E'}{(E \; [\!] \; F) \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{(E \; [\!] \; F) \xrightarrow{a} F'}$$

Seq $$\frac{E \xrightarrow{a} E'}{(E; F) \xrightarrow{a} (E'; F)}$$

Iter $$\frac{((E); *E) \xrightarrow{a} E'}{*E \xrightarrow{a} E'}$$

We extend $\xrightarrow{a}$ to processes by stipulating that $P_i \xrightarrow{a} P_i'$ iff $\mathit{body}(P_i) \xrightarrow{a} \mathit{body}(P_i')$ and $\mathit{alphabet}(P_i) = \mathit{alphabet}(P_i')$. In other words, the alphabets are the same and the bodies are related by $\xrightarrow{a}$. Finally, we extend $\xrightarrow{a}$ to programs as follows:

Let $P = (\| \; i \in \varphi : P_i)$, $P' = (\| \; i \in \varphi : P_i')$. Then $P \xrightarrow{a} P'$ iff:

1. *for all* $i \in PA_P(a) : P_i \xrightarrow{a} P_i'$

2. *for all* $i \in \varphi - PA_P(a) : P_i = P_i'$

We write $P \xrightarrow{a}$ to mean that there exists a P' such that $P \xrightarrow{a} P'$. In this case, we say that a is *enabled* in P. We also write $P \overset{a}{\not\rightarrow}$ to mean that there does not exist a P' such that $P \xrightarrow{a} P'$, and we say that a is *disabled* in P in this case.

Suppose $P_i \xrightarrow{c}$. Then the general form for the body of P_i is $F; *E$, where F has one of the forms c, $c; G$, $c \parallel H$, $c; G \parallel H$. All of these forms are subsumed by the form $c; G \parallel H$ however, since $c = c; \varepsilon \parallel 0$, $c; G = c; G \parallel 0$, $c \parallel G = c; \varepsilon \parallel G$. Thus the introduction of 0 and ε allows us to avoid a large amount of tedious case-analysis. We now present some preliminary definitions and results.

Definition 8 (*Derivative, Path*) *If* $P \xrightarrow{a_1} \cdots \xrightarrow{a_n} P'$ *for some sequence* $a_1, \ldots, a_n$ *of actions, then (again following [Milner 89]) we say that* P' *is a* derivative *of* P*. The sequence* $a_1, \ldots, a_n$ *is called a* path*. If path* $\pi = a_1, \ldots, a_n$*, then we abbreviate* $P \xrightarrow{a_1} \cdots \xrightarrow{a_n} P'$ *by* $P \xrightarrow{\pi} P'$*.*

Consider a program consisting of a single process $P_1 = *[a; b \parallel a; c]$. Clearly, $P_1 \xrightarrow{a} (b; P_1)$, and $P_1 \xrightarrow{a} (c; P_1)$. This example can easily be extended to arbitrary paths. Thus, establishing $P \xrightarrow{\pi} P'$ and $P \xrightarrow{\pi} P''$ for some P, π, P', P'', does not allow us to conclude $P' = P''$. Thus, if P and π are given, then the assertion $P \xrightarrow{\pi} P'$ can be regarded as an abbreviation for "let $P \xrightarrow{\pi} P'$ for some P'."

Suppose we have a path $\pi = \pi' bc \pi''$. Then, the path $\pi' cb \pi''$ is said to be obtained from π by a single *exchange* of actions b and c.

Definition 9 (*Independent*) *Two actions* b, c *are* independent *in program* P *iff* $PA_P(b) \cap PA_P(c) = \emptyset$

Definition 10 (*Equivalent*) *Two paths* π, ρ *are* equivalent *iff one can be obtained from the other by a finite number of exchanges of adjacent independent actions.*

Proposition 1 *If actions* b, c *are independent in program* P*, and* $P \xrightarrow{bc} P'$*, then* $P \xrightarrow{cb} P'$*.*

Proof: Since b and c are independent in P, we have $PA_P(b) \cap PA_P(c) = \emptyset$ by definition 9. Let $P = (\parallel i \in \varphi : P_i)$, and P'' be such that $P \xrightarrow{b} P'' \xrightarrow{c} P'$. Then, by definition 7,

$$P'' = (\parallel i \in PA_P(b) : P_i'') \parallel (\parallel i \in \varphi - PA_P(b) : P_i), \quad \text{where } P_i \xrightarrow{b} P_i'' \text{ for all } i \in PA_P(b)$$

Hence, by definition 7 and $PA_P(b) \cap PA_P(c) = \emptyset$,

$$P' = (\parallel i \in PA_P(b) : P_i'') \parallel (\parallel i \in PA_P(c) : P_i') \parallel (\parallel i \in \varphi - (PA_P(b) \cup PA_P(c)) : P_i),$$

where $P_i \xrightarrow{c} P_i'$ for all $i \in PA_P(c)$

By $P_i \xrightarrow{c} P'_i$ for all $i \in PA_P(c)$ and definition 7,

$$P \xrightarrow{c} P''' \text{ where } P''' = (\| \, i \in PA_P(c) : P'_i) \; \| \; (\| \, i \in \varphi - PA_P(c) : P_i)$$

Then, by $P_i \xrightarrow{b} P''_i$ for all $i \in PA_P(b)$ and definition 7,

$$P''' \xrightarrow{b} (\| \, i \in PA_P(c) : P'_i) \; \| \; (\| \, i \in PA_P(b) : P''_i) \; \| \; (\| \, i \in \varphi - (PA_P(b) \cup PA_P(c)) : P_i)$$

Hence $P''' \xrightarrow{b} P'$, and so $P \xrightarrow{c} P''' \xrightarrow{b} P'$. Thus $P \xrightarrow{cb} P'$. □

Proposition 2 *Let $P \xrightarrow{\pi} Q$. If π and ρ are equivalent, then $P \xrightarrow{\rho} Q$.*

Proof: The proof is by induction on the number m of exchanges of independent adjacent actions required to obtain ρ from π.

Base Case: $m = 1$.

Now ρ is obtained from π by one exchange. Hence we can write $\pi = \pi' ab \pi''$, $\rho = \pi' ba \pi''$, where a, b are the exchanged independent actions. Thus we have

$$P \xrightarrow{\pi'} P' \xrightarrow{ab} P'' \xrightarrow{\pi''} Q \qquad (*)$$

for some P', P''.

Since a, b are independent in P, we can apply proposition 1 to $P' \xrightarrow{ab} P''$, thereby concluding $P' \xrightarrow{ba} P''$. Using this result and (*) we have $P \xrightarrow{\pi'} P' \xrightarrow{ba} P'' \xrightarrow{\pi''} Q$. Hence $P \xrightarrow{\rho} Q$. Thus the base case is established.

Induction Step: $m = n + 1$, $n \geq 1$, where the inductive hypothesis is assumed for n exchanges.

Since ρ is obtained from π by $n+1$ exchanges, there must exist a η such that η is obtained from π by n exchanges, and ρ is obtained from η by one exchange. By the inductive hypothesis, we have $P \xrightarrow{\eta} Q$. Since ρ is obtained from η by one exchange, we use same argument as employed in the base case (i.e., for a single exchange) to conclude $P \xrightarrow{\rho} Q$. This establishes the induction step. □

3 The Right-sequence Introduction Transformation

The right-sequence introduction transformation allows us to introduce a new action, d, in sequence with, and immediately after, an already-present action, c. Intuitively, we use such a transformation to refine c. In the original high-level program, c might model a complex set of activities. In the transformed (lower-level) program, this set of activities is split between c and d.

Definition 11 (*Right-sequence Introduction Transformation*)

We define the right-sequence introduction transformation $[c/c; d]$ *in a bottom-up manner as follows. Let a be an arbitrary action, and E, F be arbitrary action expressions. Then, we have*

$$\varepsilon[c/c; d] = \varepsilon$$

$0[c/c;d] = 0$

$a[c/c;d] = a$ *if* $a \neq c$

$c[c/c;d] = c; d$

$(E \mathbin{[\!]} F)[c/c;d] = ((E[c/c;d]) \mathbin{[\!]} (F[c/c;d]))$

$(E;F)[c/c;d] = ((E[c/c;d]);(F[c/c;d]))$

In the sequel, we will use the abbreviation E_t *for* $E[c/c;d]$ *for an arbitrary action expression* E.

For an arbitrary process P_i *such that* $c \in alphabet(P_i)$, *and* $body(P_i) = F; *E$ *for some action expressions* F, E, *define* $P_i[c/c;d] = Q_i$, *where* $alphabet(Q_i) = alphabet(P_i) \cup \{d\}$, $body(Q_i) = F_t; *E_t$.

Let $P = (\ \| \ i \in \varphi : P_i)$ *be an arbitrary program. Let* ψ *be an arbitrary nonempty subset of* $PA_P(c)$. *We define* $P[c/c;d] = (\ \| \ i \in \psi : P_i[c/c;d]) \ \| \ (\ \| \ i \in \varphi - \psi : P_i)$.

If $Q = P[c/c;d]$, then we say that Q results from P by means of a *right-sequence introduction transformation.* The right-sequence introduction transformation $[c/c;d]$ takes a program P containing an action c, and introduces a new action d after c and in sequence with c. Note that ψ is an implicit parameter of the functional mapping from P to Q expressed by $Q = P[c/c;d]$. In the sequel, whenever we write $Q = P[c/c;d]$, we shall implicitly assume that the conditions $alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$ are all true.

The basic idea is that the action c in program P is decomposed into the sequence of c followed by d in program Q. The transformation can be iterated any number of times, so that c is decomposed into a sequence $c, d_1, \ldots, d_n$. Having defined the right-sequence introduction transformation, we need to relate the behavior of the transformed program $Q = P[c/c;d]$ to that of the original program P. We do this by means of a modification of the notion of *strong bisimulation* (see [Milner 89], ch. 4).

Definition 12 (*cd-bisimulation*)

Let $\mathcal{S}$ *be a binary relation over programs. Then* $\mathcal{S}$ *is a cd-*bisimulation *iff* $P \, \mathcal{S} \, Q$ *implies:*

1. $alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$
2. *for all* $a \in alphabet(P) - \{c\}$, *if* $P \xrightarrow{a} P'$, *then* $Q \xrightarrow{a} Q'$ *for some* Q' *such that* $P' \, \mathcal{S} \, Q'$
3. *if* $P \xrightarrow{c} P'$, *then* $Q \xrightarrow{cd} Q'$ *for some* Q' *such that* $P' \, \mathcal{S} \, Q'$
4. *for all* $a \in alphabet(P) - \{c\}$, *if* $Q \xrightarrow{a} Q'$, *then* $P \xrightarrow{a} P'$ *for some* P' *such that* $P' \, \mathcal{S} \, Q'$
5. *if* $Q \xrightarrow{cd} Q'$, *then* $P \xrightarrow{c} P'$ *for some* P' *such that* $P' \, \mathcal{S} \, Q'$

Definition 13 *Following the treatment of strong bisimulation given in chapter 4 of [Milner 89], we define*

$$\sim \ \overset{\mathrm{df}}{=} \bigcup \{ \mathcal{S} \mid \mathcal{S} \text{ is a cd-bisimulation} \}.$$

Proposition 3 *$P \sim Q$ iff*

1. *$alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$*
2. *for all $a \in alphabet(P) - \{c\}$, if $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ for some Q' such that $P' \sim Q'$*
3. *if $P \xrightarrow{c} P'$, then $Q \xrightarrow{cd} Q'$ for some Q' such that $P' \sim Q'$*
4. *for all $a \in alphabet(P) - \{c\}$, if $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ for some P' such that $P' \sim Q'$*
5. *if $Q \xrightarrow{cd} Q'$, then $P \xrightarrow{c} P'$ for some P' such that $P' \sim Q'$*

Proof: The proof proceeds in the same way as the proof of proposition 4 in chapter 4 of [Milner 89] (bearing in mind that the notions of bisimulation differ technically). We paraphrase the proof here for completeness.

First, we define the relation $\sim'$ as follows:

$P \sim' Q$ iff

1. $alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$
2. for all $a \in alphabet(P) - \{c\}$, if $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ for some Q' such that $P' \sim Q'$
3. if $P \xrightarrow{c} P'$, then $Q \xrightarrow{cd} Q'$ for some Q' such that $P' \sim Q'$
4. for all $a \in alphabet(P) - \{c\}$, if $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ for some P' such that $P' \sim Q'$
5. if $Q \xrightarrow{cd} Q'$, then $P \xrightarrow{c} P'$ for some P' such that $P' \sim Q'$

From definition 13, we see that $\sim$ is a *cd*-bisimulation. Thus from definition 12 and the definition of $\sim'$, we deduce:

$P \sim Q$ implies $P \sim' Q$ (P1)

We now show

$\sim'$ is a *cd*-bisimulation (P2)

Proof of P2.

Let $P \sim' Q$. If we establish all the clauses of definition 12 (for $\mathcal{S} = \sim'$), then, by that definition, we can conclude that $\sim'$ is a *cd*-bisimulation. Clause 1 follows from (and is identical to) clause 1 of the definition of $\sim'$. For clause 2, let $a \in alphabet(P) - \{c\}$ and $P \xrightarrow{a} P'$. By the definition of $\sim'$, there exists Q' such that $Q \xrightarrow{a} Q'$ and $P' \sim Q'$. From $P' \sim Q'$ and P1, we have $P' \sim' Q'$. Since $Q \xrightarrow{a} Q'$ and $P' \sim' Q'$, clause 2 of definition 12 (for $\mathcal{S} = \sim'$) is satisfied. Clauses 3–5 are verified in an identical manner.

(end proof of P2).

Since $\sim'$ is a *cd*-bisimulation, and $\sim$ is the largest *cd*-bisimulation, we have $P \sim' Q$ implies $P \sim Q$. From this and P1, we get $P \sim' Q$ iff $P \sim Q$. Replacing $P \sim' Q$ by $P \sim Q$ in the definition of $\sim'$ then gives us proposition 3. $\square$

Proposition 4 *Let $\mathcal{S} = \{(P,Q) \mid Q = P[c/c;d]\}$, then $\mathcal{S}$ is a cd-bisimulation.*

Proof: We establish each clause of definition 12 in turn. Let $P = (\| \; i \in \varphi : P_i), Q = (\| \; i \in \varphi : Q_i)$. Let G be an action expression. We shall use G_t for G with all occurrences of c substituted with $c; d$.

Clause 1: $\mathit{alphabet}(P) \cup \{d\} = \mathit{alphabet}(Q)$, $d \notin \mathit{alphabet}(P)$, $c \in \mathit{alphabet}(P)$, $c \in \mathit{alphabet}(Q)$.

The assumption we make on the alphabets of P, Q whenever we write $Q = P[c/c;d]$ (see page 7), is identical to this clause. Hence clause 1 holds by assumption.

Clause 2: for all $a \in \mathit{alphabet}(P) - \{c\}$, if $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ for some Q' such that $P' \, \mathcal{S} \, Q'$

Let $P \xrightarrow{a} P'$ for some $P' = (\| \; i \in \varphi : P_i')$ and $a \in \mathit{alphabet}(P) - \{c\}$. We show that there exists a $Q' = (\| \; i \in \varphi : Q_i')$ such that $Q \xrightarrow{a} Q'$ and $P' \, \mathcal{S} \, Q'$. By $P \xrightarrow{a} P'$ and the transition relation definition (7), $P_i \xrightarrow{a} P_i'$ for all $i \in PA_P(a)$, and $P_i = P_i'$ for all $i \in \varphi - PA_P(a)$. We have two cases.

Case 1: $i \in PA_Q(a)$.

Consider P_i, Q_i, P_i' for an arbitrary $i \in PA_Q(a)$. Since $P_i \xrightarrow{a}$, $\mathit{body}(P_i) = (a; F \; [] \; G); *E$ as discussed above (page 5), for some action expressions F, G, E.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((a; F \; [] \; G); *E) \;\xrightarrow{a}\; (F; *E)$$

In general, P_i can have more than one a-derivative, since G itself could have the form $a; F' \; [] \; G'$. However, there is no loss of generality in assuming that the execution of the a in $a; F$ leads to P_i'. Hence we have $\mathit{body}(P_i') = (F; *E)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have two subcases, $i \in PA_Q(a) \cap PA_Q(d)$ and $i \in PA_Q(a) - PA_Q(d)$.

Subcase 1.1: $i \in PA_Q(a) \cap PA_Q(d)$.

From $Q = P[c/c;d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i[c/c;d]$. Since $\mathit{body}(P_i) = (a; F \; [] \; G); *E$, we get $\mathit{body}(Q_i) = (a; F_t \; [] \; G_t); *E_t$, by the right-sequence introduction transformation (11). By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$$((a; F_t \; [] \; G_t); *E_t) \;\xrightarrow{a}\; (F_t; *E_t)$$

Since $\mathit{body}(Q_i) = (a; F_t \; [] \; G_t); *E_t$, there exists a Q_i' such that $Q_i \xrightarrow{a} Q_i'$, and $\mathit{body}(Q_i') = (F_t; *E_t)$. Since $\mathit{body}(P_i') = F; *E$, we have $Q_i' = P_i'[c/c;d]$ by the right-sequence introduction transformation (11).

Subcase 1.2: $i \in PA_Q(a) - PA_Q(d)$.

From $Q = P[c/c;d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i$. Letting Q_i' be P_i', we get $Q_i \xrightarrow{a} Q_i'$ and $Q_i = P_i$, since $P_i \xrightarrow{a} P_i'$.

Case 2: $i \in \varphi - PA_Q(a)$.

Consider P_i, Q_i, P_i' for an arbitrary $i \in \varphi - PA_Q(a)$. As before, by $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have two subcases, $i \in (\varphi - PA_Q(a)) - PA_Q(d)$ and $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$

Subcase 2.1: $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

From $Q = P[c/c;d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i$. Therefore $P_i' = Q_i'$ since neither of P_i, Q_i participate in the action a (and so $P_i = P_i', Q_i = Q_i'$).

Subcase 2.2: $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$.

From $Q = P[c/c;d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i[c/c;d]$. Again neither of P_i, Q_i participate in a, and so $P_i' = P_i$ and $Q_i' = Q_i$. Hence, we have $Q_i' = P_i'[c/c;d]$.

If we now consider $Q' = (\| \ i \in \varphi : Q_i')$, where the Q_i' are as given by the preceding case analysis, we see that $Q \xrightarrow{a} Q'$ by the transition relation definition (7), and also that $Q' = P'[c/c;d]$, by the right-sequence introduction transformation definition (11), since the above two cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P,Q) \mid Q = P[c/c;d]\}$, we have $P' \, \mathcal{S} \, Q'$ as required.

<u>Clause 3: if $P \xrightarrow{c} P'$, then $Q \xrightarrow{cd} Q'$ for some Q' such that $P' \, \mathcal{S} \, Q'$</u>

Let $P \xrightarrow{c} P'$ for some $P' = (\| \ i \in \varphi : P_i')$. We show that there exists a $Q' = (\| \ i \in \varphi : Q_i')$ such that $Q \xrightarrow{cd} Q'$ and $P' \, \mathcal{S} \, Q'$. By $P \xrightarrow{c} P'$ and the transition relation definition (7), $P_i \xrightarrow{c} P_i'$ for all $i \in PA_P(c)$, and $P_i = P_i'$ for all $i \in \varphi - PA_P(c)$. We have three cases.

Case 1: $i \in PA_Q(d)$.

Consider P_i, Q_i, P_i' for an arbitrary $i \in PA_Q(d)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have $Q_i = P_i[c/c;d]$. Since $P_i \xrightarrow{c}$, we have $body(P_i) = (c; F \parallel G); *E$ as discussed above (page 5), for some action expressions F, G, E.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((c; F \parallel G); *E) \ \xrightarrow{c} \ (F; *E)$$

In general, P_i can have more than one c-derivative, since G itself could have the form $c; F' \parallel G'$. However, there is no loss of generality in assuming that the execution of the c in $c; F$ leads to P_i'. Hence we have $body(P_i') = F; *E$. Now $Q_i = P_i[c/c;d]$. Since $body(P_i) = (c; F \parallel G); *E$, we get $body(Q_i) = (c; d; F_t \parallel G_t); *E_t$, by the right-sequence introduction transformation (11). By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((c; d; F_t \parallel G_t); *E_t) \ \xrightarrow{c} \ (d; F_t; *E_t)$$

By rules **Act, Seq**, we have

$$(d; F_t; *E) \ \xrightarrow{d} \ (F_t; *E_t)$$

Hence, we conclude

$$((c;d;F_t \,[\!]\, G_t);*E_t) \overset{cd}{\rightarrow} (F_t;*E_t)$$

Since $body(Q_i) = (c;d;F_t \,[\!]\, G_t);*E_t$, there exists a Q'_i such that $Q_i \overset{cd}{\rightarrow} Q'_i$, and $body(Q'_i) = F_t;*E_t$. Since $body(P'_i) = F;*E$, we have $Q'_i = P'_i[c/c;d]$ by the right-sequence introduction transformation (11).

Case 2: $i \in PA_Q(c) - PA_Q(d)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(c) - PA_Q(d)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have $Q_i = P_i$. Letting Q'_i be P'_i, we get $Q_i \overset{c}{\rightarrow} Q'_i$ and $Q'_i = P'_i$.

Case 3: $i \in \varphi - PA_Q(c)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in \varphi - PA_Q(c)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have $Q_i = P_i$. Since $P_i = P'_i$ in this case, we let $Q'_i = Q_i$, and hence $Q'_i = P'_i$.

If we now consider $Q' = (\| \ i \in \varphi : Q'_i)$, where the Q'_i are as given by the preceding case analysis, we see that $Q \overset{cd}{\rightarrow} Q'$ by the transition relation definition (7), and also that $Q' = P'[c/c;d]$, by the right-sequence introduction transformation definition (11), since the above three cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P,Q) \mid Q = P[c/c;d]\}$, we have $P' \,\mathcal{S}\, Q'$ as required.

<u>Clause 4: for all $a \in alphabet(P) - \{c\}$, if $Q \overset{a}{\rightarrow} Q'$, then $P \overset{a}{\rightarrow} P'$ for some P' such that $P' \,\mathcal{S}\, Q'$</u>

Let $Q \overset{a}{\rightarrow} Q'$ for some $Q' = (\| \ i \in \varphi : Q'_i)$ and $a \in alphabet(P) - \{c\}$. We show that there exists a $P' = (\| \ i \in \varphi : P'_i)$ such that $P \overset{a}{\rightarrow} P'$ and $P' \,\mathcal{S}\, Q'$. By $Q \overset{a}{\rightarrow} Q'$ and the transition relation definition (7), $Q_i \overset{a}{\rightarrow} Q'_i$ for all $i \in PA_Q(a)$, and $Q_i = Q'_i$ for all $i \in \varphi - PA_Q(a)$. We have two cases.

Case 1: $i \in PA_Q(a)$.

Consider P_i, Q_i, Q'_i for an arbitrary $i \in PA_Q(a)$. From $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11) we have two subcases, $i \in PA_Q(a) \cap PA_Q(d)$ and $i \in PA_Q(a) - PA_Q(d)$.

Subcase 1.1: $i \in PA_Q(a) \cap PA_Q(d)$.

From $Q = P[c/c;d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i[c/c;d]$. Since $Q_i \overset{a}{\rightarrow}$, we have $body(Q_i) = (a;F' \,[\!]\, G');*E'$ for some action expressions F', G', E', as discussed above (page 5). From $Q_i = P_i[c/c;d]$ and the right-sequence introduction transformation definition (11), we conclude that $body(P_i) = (a;F \,[\!]\, G);*E$ where action expressions E, F, G are such that $F' = F_t, G' = G_t, E' = E_t$.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((a;F_t \,[\!]\, G_t);*E_t) \overset{a}{\rightarrow} (F_t;*E_t)$$

In general, Q_i can have more than one a-derivative, since G_t itself could have the form $a;F'' \,[\!]\, G''$. However, there is no loss of generality in assuming that the execution of the a in $a;F_t$ leads to Q'_i. Hence we have

$body(Q'_i) = (F_t; *E_t)$.

Now $body(P_i) = (a; F \,[\!]\, G); *E$. By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$((a; F \,[\!]\, G); *E) \ \overset{a}{\to}\ (F; *E)$

Since $body(P_i) = (a; F \,[\!]\, G); *E$, there exists a P'_i such that $P_i \overset{a}{\to} P'_i$ and $body(P'_i) = (F; *E)$. Since $body(Q'_i) = F_t; *E_t$, we have $Q'_i = P'_i[c/c; d]$ by the right-sequence introduction transformation (11).

Subcase 1.2: $i \in PA_Q(a) - PA_Q(d)$.

From $Q = P[c/c; d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i$. Letting P'_i be Q'_i, we get $P_i \overset{a}{\to} P'_i$ and $P'_i = Q'_i$.

Case 2: $i \in \varphi - PA_Q(a)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in \varphi - PA_Q(a)$. By $Q = P[c/c; d]$ and the right-sequence introduction transformation definition (11), we have two subcases, $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$ and $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

Subcase 2.1: $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$.

From $Q = P[c/c; d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i[c/c; d]$. Since $i \in \varphi - PA_Q(a)$ these Q_i do not participate in action a. Therefore we have $Q'_i = Q_i$. We also have $P'_i = P_i$ since these P_i do not participate in action a either. Since $Q_i = P_i[c/c; d]$ we have $Q'_i = P'_i[c/c; d]$

Subcase 2.2: $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

From $Q = P[c/c; d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i$. Since $i \in \varphi - PA_Q(a)$ these Q_i do not participate in action a. Letting P'_i be Q'_i, we have $Q_i = P_i = Q'_i = P'_i$.

If we now consider $P' = (\| \ i \in \varphi : P'_i)$, where the P'_i are as given by the preceding case analysis, we see that $P \overset{a}{\to} P'$ by the transition relation definition (7), and also that $Q' = P'[c/c; d]$, by the right-sequence introduction transformation definition (11), since the above two cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P, Q) \mid Q = P[c/c; d]\}$, we have $P' \,\mathcal{S}\, Q'$ as required.

Clause 5: if $Q \overset{cd}{\to} Q'$, then $P \overset{c}{\to} P'$ for some P' such that $P' \,\mathcal{S}\, Q'$

Let $Q \overset{cd}{\to} Q'$ for some $Q' = (\| \ i \in \varphi : Q'_i)$. We show that there exists a $P' = (\| \ i \in \varphi : P'_i)$ such that $P \overset{c}{\to} P'$ and $P' \,\mathcal{S}\, Q'$. By $Q \overset{cd}{\to} Q'$ and the transition relation definition (7), $Q_i \overset{cd}{\to} Q'_i$ for all $i \in PA_Q(d)$, $Q_i \overset{c}{\to} Q'_i$ for all $i \in PA_Q(c) - PA_Q(d)$ and $Q_i = Q'_i$ for all $i \in \varphi - PA_Q(c)$. We have three cases.

Case 1: $i \in PA_Q(d)$

Consider Q_i, P_i, Q'_i for an arbitrary $i \in PA_Q(d)$. From $Q = P[c/c; d]$, the right-sequence introduction transformation definition (11), and the subcase condition, we have $Q_i = P_i[c/c; d]$. Hence, d cannot occur as an operand of $[\!]$ in Q_i. From this and $Q_i \overset{cd}{\to} Q'_i$, we have $body(Q_i) = (c; d; F' \,[\!]\, G'); *E'$ for some action expressions F', G', E', as discussed above (page 5). From $Q_i = P_i[c/c;d]$ and the right-sequence introduction transformation definition (11), we conclude that $body(P_i) = (c;F \,[\!]\, G);*E$ where action expressions E, F, G are such that $F' = F_t, G' = G_t, E' = E_t$. By the transition relation definition (7), rule **Act**, we have

$$((c;d;F_t \,[\!]\, G_t);*E_t) \xrightarrow{c} (d;F_t;*E_t)$$

By rule **Act**, we have

$$(d;F_t;*E_t) \xrightarrow{d} (F_t;*E_t)$$

Hence,

$$(c;d;F_t \,[\!]\, G_t);*E_t \xrightarrow{cd} (F_t;*E_t)$$

In general, Q_i can have more than one cd-derivative, since G_t itself could have the form $c;d;F'' \,[\!]\, G''$. However, there is no loss of generality in assuming that the execution of the c and then d in $c;d;F_t$ leads to Q_i'. Hence we have $body(Q_i') = (F_t;*E_t)$.

Now $body(P_i) = (c;F \,[\!]\, G);*E$. By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$$((c;F \,[\!]\, G);*E) \xrightarrow{c} (F;*E)$$

Since $body(P_i) = (c;F \,[\!]\, G);*E$, there exists a P_i' such that $P_i \xrightarrow{c} P_i'$ and $body(P_i') = F;*E$. Since $body(Q_i') = F_t;*E_t$, we have $Q_i' = P_i'[c/c;d]$ by the right-sequence introduction transformation (11).

Case 2: $i \in PA_Q(c) - PA_Q(d)$.

Consider P_i, Q_i, Q_i' for an arbitrary $i \in PA_Q(c) - PA_Q(d)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have $Q_i = P_i$. Letting P_i' be Q_i', we get $P \xrightarrow{c} P'$ since $Q \xrightarrow{c} Q'$.

Case 3: $i \in \varphi - PA_Q(c)$.

Consider Q_i, P_i, Q_i' for an arbitrary $i \in \varphi - PA_Q(c)$. By $Q = P[c/c;d]$ and the right-sequence introduction transformation definition (11), we have $Q_i = P_i$. Now $Q_i = Q_i'$ in this case as explained above. Letting $P_i' = P_i$ we get $P_i' = Q_i'$.

If we now consider $P' = (\| \; i \in \varphi : P_i')$, where the P_i' are as given by the preceding case analysis, we see that $P \xrightarrow{c} P'$ by the transition relation definition (7), and also that $Q' = P'[c/c;d]$, by the right-sequence introduction transformation definition (11), since the above three cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P,Q) \mid Q = P[c/c;d]\}$, we have $P' \,\mathcal{S}\, Q'$ as required.

We have shown that all five clauses of definition 12 hold, hence $\mathcal{S}$ is a cd-bisimulation. □

Proposition 5 *Let $Q = P[c/c;d]$. If Q' is an arbitrary derivative of Q, then either $Q' \xrightarrow{d}$, or there exists a derivative P' of P such that $P' \sim Q'$.*

Proof: Let $\mathcal{S} = \{(P,Q) \mid Q = P[c/c;d]\}$. By proposition 4, $\mathcal{S}$ is a cd-bisimulation. Hence $\mathcal{S} \subseteq \sim$ by definition 13. Since $P \,\mathcal{S}\, Q$ by assumption, we conclude $P \sim Q$. Since Q' is a derivative of Q, we have $Q \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q'$ for some path $\pi = a_1, \ldots, a_n$ of length n. Now π will contain some number (possibly 0) of occurrences of c. Furthermore, from the syntactic form of Q and the transition relation (definition 7), we have that every pair of successive occurrences of c along π have exactly one occurrence of d between them, and likewise every pair of successive occurrences of d along π have exactly one occurrence of c between them. Furthermore, the first occurrence of c in π (if any) is not preceded by an occurrence of d.

Suppose that d does not occur in the suffix of π starting with the last occurrence of c in π. Then, from the syntactic form of Q, the transition relation (definition 7), and $Q \xrightarrow{\pi} Q'$, we conclude $Q' \overset{d}{\not\to}$, and we are done. Hence we can assume in the rest of the proof that there is exactly one occurrence of d in the suffix of π starting with the last occurrence of c in π. In other words, every occurrence of c can be matched with the subsequent occurrence of d. Consider a segment of π starting with some arbitrary occurrence of c in π and ending with the matching occurrence of d. This segment has the form $c, b_1, \ldots, b_m, d$ where none of $b_1, \ldots, b_n$ is either c or d.

From $Q = P[c/c; d]$, we have that any $Q_i \in PA_d(Q)$ which executes c can only execute d as its next action; it has no other choice. Thus we conclude that no $Q_i \in PA_d(Q)$ participates in any of the actions $b_2, b_2, \ldots, b_m$. Thus, by definition 9, d and b_j are independent, for $1 \leq j \leq m$. Hence, by definition 10, the paths $c, b_1, b_2, \ldots, b_m, d$ and $c, d, b_1, b_2, \ldots, b_m$ are equivalent paths. Let ρ be the path obtained from π by replacing every segment of the form $c, b_1, \ldots, b_m, d$ by the segment $c, d, b_1, b_2, \ldots, b_m$. From the definition of equivalence (def. 10), we easily see that equivalence is preserved by path-concatenation. Hence ρ is equivalent to π. Thus, by proposition 2 and $Q \xrightarrow{\pi} Q'$, we have $Q \xrightarrow{\rho} Q'$. We now prove:

there exists a derivative P' of P such that $P' \sim Q'$ (P1)

Proof of P1.

The proof is by induction on the length ℓ of ρ.

Base Case, $\ell = 0$

Hence ρ is the empty path, and so $Q' = Q$. Let P' be P. By assumption, $Q = P[c/c; d]$. Thus by proposition 4, $P \sim Q$. Hence $P' \sim Q'$.

Induction Step, $\ell = n + 1$

We assume the inductive hypothesis for $\ell \leq n$ $(n \geq 0)$ and establish P1 for $\ell = n + 1$. Since ρ has length $n + 1$, it can be written as $a_1, \ldots, a_n, a_{n+1}$. Since every c in ρ is immediately followed by a d in ρ, a_{n+1} cannot be c. Thus we have two cases.

Case 1: $a_{n+1} = d$.

Hence $a_n = c$. Thus $\rho = a_1, \ldots, c, d$. From this and $Q \xrightarrow{\rho} Q'$, there is a Q'' be such that $Q \overset{a_1}{\to} \cdots \overset{a_{n-1}}{\to} Q''$ and $Q'' \overset{cd}{\to} Q'$. Assuming the inductive hypothesis for $\ell = n - 1$, we have that there exists a derivative P'' of P such that $P'' \sim Q''$. From this, $Q'' \overset{cd}{\to} Q'$, and proposition 3, we have $P'' \overset{c}{\to} P'$ for some P' such that $P' \sim Q'$. Since P' is a derivative of P, P1 is established in this case.

Case 2: $a_{n+1} \neq d$.

From $\rho = a_1, \ldots, a_n, a_{n+1}$ and $Q \xrightarrow{\rho} Q'$, there is a Q'' be such that $Q \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q''$ and $Q'' \xrightarrow{a_{n+1}} Q'$. Assuming the inductive hypothesis for $\ell = n$, we have that there exists a derivative P'' of P such that $P'' \sim Q''$. From this, $Q'' \xrightarrow{a_{n+1}} Q'$, and proposition 3, we have $P'' \xrightarrow{a_{n+1}} P'$ for some P' such that $P' \sim Q'$. Since P' is a derivative of P, P1 is established in this case, which completes the induction step.

(end proof of P1)

Since P1 implies the proposition, we are done. □

As stated in the introduction, our aim is to design syntactic transformations which automatically preserve desirable program properties. We now show that the right-sequence introduction transformation preserves deadlock-freedom.

Definition 14 (*Deadlock-Freedom*) *If for every derivative P' of P, there is some action a such that $P' \xrightarrow{a}$, then P is deadlock-free.*

Theorem 6 *Let $Q = P[c/c; d]$. If P is deadlock-free, then so is Q.*

Proof: Let Q' be an arbitrary derivative of Q. By definition 14, it suffices to show $Q' \xrightarrow{a}$ for some action a. From proposition 4 and $Q = P[c/c; d]$, we have $P \sim Q$. Thus, by proposition 5, we have

either $Q' \xrightarrow{d}$, or there exists a derivative P' of P such that $P' \sim Q'$

If $Q' \xrightarrow{d}$ then we are done. Otherwise, there exists a derivative P' of P such that $P' \sim Q'$. By assumption, P is deadlock-free. Hence, by definition 14, $P' \xrightarrow{a'}$ for some a'. If $a' \neq c$, then $Q' \xrightarrow{a'}$ by proposition 3. If $a' = c$, then $Q' \xrightarrow{cd}$, again by proposition 3. Hence in all cases we have $Q' \xrightarrow{a}$ for some a. □

4 The Left-Sequence Introduction Transformation

The left-sequence introduction transformation allows us to introduce a new action, d, in sequence with, and immediately before, an already-present action, c. This second transformation gives us greater choice and latitude in carrying out refinements.

Definition 15 (*Left-sequence Introduction Transformation*)

We define the left-sequence introduction transformation $[c/d; c]$ *in a bottom-up manner as follows. Let a be an arbitrary action, and E, F be arbitrary action expressions. Then, we have*

$\varepsilon[c/d; c] = \varepsilon$

$0[c/d; c] = 0$

$a[c/d; c] = a$ *if* $a \neq c$

$c[c/d; c] = d; c$

$(E \,[\!]\, F)[c/d;c] = ((E[c/d;c]) \,[\!]\, (F[c/d;c]))$

$(E;F)[c/d;c] = ((E[c/d;c]);(F[c/d;c]))$

In the sequel, we will use the abbreviation E_t for $E[c/d;c]$ for an arbitrary action expression E.

For an arbitrary process P_i such that $c \in alphabet(P_i)$, and $body(P_i) = F;*E$ for some action expressions F, E, define $P_i[c/d;c] = Q_i$, where $alphabet(Q_i) = alphabet(P_i) \cup \{d\}$, $body(Q_i) = F_t;*E_t$. In addition, if $body(P_i) = c;F;*E$, for some action expressions F, E, we define $P_i[c/d;c,nl] = Q_i$, where $alphabet(Q_i) = alphabet(P_i) \cup \{d\}$, $body(Q_i) = c;F_t;*E_t$, i.e., the leading c action is not preceded by a d action (this is needed for technical reasons).

Let $P = (\,\|\; i \in \varphi : P_i)$ be an arbitrary program such that $c \in alphabet(P)$, $d \notin alphabet(P)$ for actions c, d. Let ψ be an arbitrary nonempty subset of $PA_P(c)$ such that for all $i \in \psi$, no occurrence of c in P_i is the operand of $[\!]$. We define $P[c/d;c] = (\,\|\; i \in \psi : P_i[c/d;c]) \;\|\; (\,\|\; i \in \varphi - \psi : P_i)$. If $P \xrightarrow{c}$, then we also define $P[c/d;c,nl] = (\,\|\; i \in \psi : P_i[c/d;c,nl]) \;\|\; (\,\|\; i \in \varphi - \psi : P_i)$.

If $Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$, then we say that Q results from P by means of a *left-sequence introduction transformation.* The left-sequence introduction transformation $[c/d;c]$ takes a program P containing an action c such that some occurrences of c are not involved in a choice, and introduces a new action d before c and in sequence with c. In the sequel, whenever we write $Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$, we shall implicitly assume that the conditions $alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$ are all true.

From the transition rules we conclude that if a left-sequence introduction transformation were applied to a process body where c is in a choice with other actions, it could lead to a deadlock. For example consider the following program:

$*(e;c) \;\|\; *(e \,[\!]\, c)$

The above program is deadlock-free. Applying a left-sequence introduction transformation to this program, we obtain:

$*(e;c) \;\|\; *(e \,[\!]\, d;c)$

Any path of the form $(edc)^*d$ ends in a deadlock state. This is why we do not allow occurrences of c to be operands of $[\!]$ in the definition of the left-sequence introduction transformation. We can therefore assume the following form for the body of a process which is ready to execute c: $(c;F);*E$.

However, if a process is ready to execute an action a other than c, then a can be in a choice with other actions and therefore the same analysis as on page 5 follows. Thus if a is one of the next possible actions and $a \neq c$ then the general form for the body of a process is $(a;F \,[\!]\, G);*E$.

We now introduce the notion of dc-bisimulation, which will be used to relate programs P, Q such that $Q = P[c/d;c]$ in much the same way that cd-bisimulation was used to relate programs P, Q such that $Q = P[c/c;d]$ in the previous section.

Definition 16 (*dc-bisimulation*)

*Let $\mathcal{S}$ be a binary relation over programs. Then $\mathcal{S}$ is a dc-*bisimulation *iff $P\,\mathcal{S}\,Q$ implies:*

1. *$alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$*
2. *for all $a \in alphabet(P) - \{c\}$, if $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ for some Q' such that $P'\,\mathcal{S}\,Q'$*
3. *if $P \xrightarrow{c} P'$, then*

   either $Q \xrightarrow{dc} Q'$ for some Q' such that $P'\,\mathcal{S}\,Q'$

   or $Q \xrightarrow{c} Q'$ for some Q' such that $P'\,\mathcal{S}\,Q'$
4. *for all $a \in alphabet(P) - \{c\}$, if $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ for some P' such that $P'\,\mathcal{S}\,Q'$*
5. *if $Q \xrightarrow{d} Q'$, then $P\,\mathcal{S}\,Q'$, and*

   if $Q \xrightarrow{c} Q'$, then $P \xrightarrow{c} P'$ for some P' such that $P'\,\mathcal{S}\,Q'$

Definition 17 *Following the treatment of strong bisimulation given in chapter 4 of [Milner 89], we define*

$$\sim \stackrel{\text{df}}{=} \bigcup\{\mathcal{S} \mid \mathcal{S} \text{ is a dc-bisimulation}\}.$$

Proposition 7 *$P \sim Q$ iff*

1. *$alphabet(P) \cup \{d\} = alphabet(Q)$, $d \notin alphabet(P)$, $c \in alphabet(P)$, $c \in alphabet(Q)$*
2. *for all $a \in alphabet(P) - \{c\}$, if $P \xrightarrow{a} P'$, then $Q \xrightarrow{a} Q'$ for some Q' such that $P' \sim Q'$*
3. *if $P \xrightarrow{c} P'$, then*

   either $Q \xrightarrow{dc} Q'$ for some Q' such that $P' \sim Q'$

   or $Q \xrightarrow{c} Q'$ for some Q' such that $P' \sim Q'$
4. *for all $a \in alphabet(P) - \{c\}$, if $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ for some P' such that $P' \sim Q'$*
5. *if $Q \xrightarrow{d} Q'$, then $P \sim Q'$, and*

   if $Q \xrightarrow{c} Q'$, then $P \xrightarrow{c} P'$ for some P' such that $P' \sim Q'$

Proof: The proof proceeds in the same way as the proof of proposition 4 in chapter 4 of [Milner 89] (bearing in mind that the notions of bisimulation differ technically).

Proposition 8 *Let $\mathcal{S} = \{(P, Q) \mid Q = P[c/d; c]$ or $Q = P[c/d; c, nl]\}$. Then $\mathcal{S}$ is a dc-bisimulation.*

Proof: Our approach to this proof shall be analogous to that for proposition 4. As before we shall use P_i to represent processes of program P and P'_i to represent processes of program P'. Similarly for Q_i and Q'_i. We observe that if ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) and no occurrence of c in P is an operand of $[\!]$, then, by the left-sequence introduction transformation definition (15), no occurrence of c in Q will be an operand of $[\!]$.

$\underline{\text{Clause 1: } \mathit{alphabet}(P) \cup \{d\} = \mathit{alphabet}(Q),\ d \notin \mathit{alphabet}(P),\ c \in \mathit{alphabet}(P),\ c \in \mathit{alphabet}(Q).}$

The assumption we make on the alphabets of P, Q whenever we write ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$), (see page 16), is identical to this clause. Hence clause 1 holds by assumption.

$\underline{\text{Clause 2: for all } a \in \mathit{alphabet}(P) - \{c\}, \text{ if } P \xrightarrow{a} P', \text{ then } Q \xrightarrow{a} Q' \text{ for some } Q' \text{ such that } P' \mathcal{S} Q'}$

Let $P \xrightarrow{a} P'$ for some $P' = (\| \, i \in \varphi : P'_i)$ and $a \in \mathit{alphabet}(P) - \{c\}$. We show that there exists a $Q' = (\| \, i \in \varphi : Q'_i)$ such that $Q \xrightarrow{a} Q'$ and $P' \mathcal{S} Q'$. By $P \xrightarrow{a} P'$ and the transition relation definition (7), $P_i \xrightarrow{a} P'_i$ for all $i \in PA_P(a)$, and $P_i = P'_i$ for all $i \in \varphi - PA_P(a)$. We have two cases.

Case 1: $i \in PA_Q(a)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(a)$. Since $P_i \xrightarrow{a}$, $\mathit{body}(P_i)$ must have the form $(a; F \, [\!] \, G); *E$ as discussed above (page 16), for some action expressions F, G, E.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((a; F \, [\!] \, G); *E) \ \xrightarrow{a} \ (F; *E)$$

In general, P_i can have more than one a-derivative, since G itself could have the form $a; F' \, [\!] \, G'$. However, there is no loss of generality in assuming that the execution of the action a in $a; F$ leads to P'_i. Hence we have $\mathit{body}(P'_i) = (F; *E)$.

We have two subcases, $i \in PA_Q(a) \cap PA_Q(d)$ and $i \in PA_Q(a) - PA_Q(d)$.

Subcase 1.1: $i \in PA_Q(a) \cap PA_Q(d)$.

Now $P_i \overset{c}{\not\rightarrow}$ for all $i \in PA_P(d)$, since c is never the operand of $[\!]$ in $P_i, i \in PA_Q(d)$. From $P_i \overset{c}{\not\rightarrow}$ and definition 15, we get $Q_i \neq P_i[c/d;c,nl]$. From ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$), definition 15, and $i \in PA_Q(a) \cap PA_Q(d)$, we get ($Q_i = P_i[c/d;c]$ or $Q_i = P_i[c/d;c,nl]$). Hence $Q_i = P_i[c/d;c]$. Since $\mathit{body}(P_i) = (a; F \, [\!] \, G); *E$, we get $\mathit{body}(Q_i) = (a; F_t \, [\!] \, G_t); *E_t$, by the left-sequence introduction transformation (15). By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$$((a; F_t \, [\!] \, G_t); *E_t) \ \xrightarrow{a} \ (F_t; *E_t)$$

Since $\mathit{body}(Q_i) = (a; F_t \, [\!] \, G_t); *E_t$, there exists a Q'_i such that $Q_i \xrightarrow{a} Q'_i$, and $\mathit{body}(Q'_i) = F_t; *E_t$. Since $\mathit{body}(P'_i) = F; *E$, we have $Q'_i = P'_i[c/d;c]$ by the left-sequence introduction transformation (15).

Subcase 1.2: $i \in PA_Q(a) - PA_Q(d)$.

From ($Q \in P[c/d;c]$ or $Q \in P[c/d;c,nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have $Q_i = P_i$. Letting Q'_i be P'_i, we get $Q_i \xrightarrow{a} Q'_i$ and $Q'_i = P'_i$, since $P_i \xrightarrow{a} P'_i$.

Case 2: $i \in \varphi - PA_Q(a)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in \varphi - PA_Q(a)$. We have two subcases, $i \in (\varphi - PA_Q(a)) - PA_Q(d)$ and $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$

Subcase 2.1: $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

From ($Q = P[c/d;c]$ or $Q \in P[c/d;c,nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have $Q_i = P_i$. Therefore $P'_i = Q'_i$ since neither of P_i, Q_i participate in the action a (and so $P_i = P'_i, Q_i = Q'_i$).

Subcase 2.2: $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$.

From ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have ($Q_i = P_i[c/d;c]$ or $Q_i = P_i[c/d;c,nl]$). Again neither of P_i, Q_i participate in a, and so $P'_i = P_i$ and $Q'_i = Q_i$. Hence, we have ($Q'_i = P'_i[c/d;c]$ or $Q'_i = P'_i[c/d;c,nl]$).

Now suppose $PA_Q(a) \cap PA_Q(d) \neq \emptyset$. Then, by subcase 1.1, $Q'_i = P'_i[c/d;c]$ for some $i \in PA_Q(a) \cap PA_Q(d)$. Thus, by all the subcases above and definition 15, we conclude $Q' = P'[c/d;c]$. On the other hand, if $PA_Q(a) \cap PA_Q(d) = \emptyset$, then $(\varphi - PA_Q(a)) \cap PA_Q(d) = PA_Q(d)$, and therefore, by subcases 1.2, 2.1, 2.2, and definition 15, we conclude ($Q' = P'[c/d;c]$ or $Q' = P'[c/d;c,nl]$). Thus we have $P \mathcal{S} Q$ in all cases.

If we now consider $Q' = (\| i \in \varphi : Q'_i)$, where the Q'_i are as given by the preceding case analysis, we see that $Q \xrightarrow{a} Q'$ by the transition relation definition (7). Hence clause 2 is satisfied.

<u>Clause 3: if $P \xrightarrow{c} P'$, then either $Q \xrightarrow{dc} Q'$ for some Q' such that $P' \mathcal{S} Q'$ or $Q \xrightarrow{c} Q'$ for some Q' such that $P' \mathcal{S} Q'$</u>

Let $P \xrightarrow{c} P'$ for some $P' = (\| i \in \varphi : P'_i)$. By $P \xrightarrow{c} P'$ and the transition relation definition (7), $P_i \xrightarrow{c} P'_i$ for all $i \in PA_Q(d)$, $P_i \xrightarrow{c} P'_i$ for all $i \in PA_Q(c) - PA_Q(d)$ and $P_i = P'_i$ for all $i \in \varphi - PA_Q(c)$. We have two cases.

Case 1: $Q \xrightarrow{d}$.

We show that there exists a $Q' = (\| i \in \varphi : Q'_i)$ such that $Q \xrightarrow{dc} Q'$ and $P' \mathcal{S} Q'$.

Subcase 1.1: $i \in PA_Q(d)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(d)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$), the left-sequence introduction transformation definition (15), and $i \in PA_Q(d)$, we get ($Q_i = P_i[c/d;c]$ or $Q_i = P_i[c/d;c,nl]$). From $Q \xrightarrow{d}$, we get $Q_i \xrightarrow{d}$. Hence $Q_i \neq P_i[c/d;c,nl]$. Thus $Q_i = P_i[c/d;c]$. Since $P_i \xrightarrow{c}$ and $i \in PA_Q(d)$, $body(P_i)$ must have the form $(c;F);*E$ as discussed above (page 16).

By the transition relation definition (7), rule **Act, Seq** , we have

$$((c;F);*E) \xrightarrow{c} (F;*E)$$

Hence we have

$$body(P'_i) = (F;*E).$$

Now $Q_i = P_i[c/d;c]$. Since $body(P_i) = (c;F);*E$, we get $body(Q_i) = *(d;c;F_t);*E_t$, by the left-sequence introduction transformation (15). By the transition relation definition (7), rule **Act**, we have

$((d;c;F_t);*E_t) \xrightarrow{d} ((c;F_t);*E_t)$

By rule **Act**, we have

$(c;F_t;*E_t) \xrightarrow{c} (F_t;*E_t)$

Hence, we conclude

$((d;c;F_t);*E_t) \xrightarrow{dc} (F_t;*E_t)$

Since $body(Q_i) = ((d;c;F_t);*E_t)$, there exists a Q'_i such that $Q_i \xrightarrow{dc} Q'_i$, and $body(Q'_i) = (F_t;*E_t)$. Since $body(P'_i) = (F;*E)$, we have $Q'_i = P'_i[c/d;c]$ by the left-sequence introduction transformation (15).

Subcase 1.2: $i \in PA_Q(c) - PA_Q(d)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(c) - PA_Q(d)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Letting Q'_i be P'_i, we get $Q_i \xrightarrow{c} Q'_i$ and $Q'_i = P'_i$.

Subcase 1.3: $i \in \varphi - PA_Q(c)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in \varphi - PA_Q(c)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Since $P_i = P'_i$ in this case, we let $Q'_i = Q_i$, and hence $Q'_i = P'_i$.

If we now consider $Q' = (\| \ i \in \varphi : Q'_i)$, where the Q'_i are as given by the preceding case analysis, we see that $Q \xrightarrow{dc} Q'$ by the transition relation definition (7), and also that $Q' = P'[c/d;c]$, by the left-sequence introduction transformation definition (15), since the above three cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P,Q) \mid Q = P[c/d;c] \text{ or } Q = P[c/d;c,nl]\}$, we have $P' \, \mathcal{S} \, Q'$ as required.

Case 2: $Q \overset{d}{\not\rightarrow}$.

We show that there exists a $Q' = (\| \ i \in \varphi : Q'_i)$ such that $Q \xrightarrow{c} Q'$ and $P' \, \mathcal{S} \, Q'$. By $P \xrightarrow{c} P'$ and the transition relation definition (7), $P_i \xrightarrow{c} P'_i$ for all $i \in PA_Q(d)$, $P_i \xrightarrow{c} P'_i$ for all $i \in PA_Q(c) - PA_Q(d)$, $P_i = P'_i$ for all $i \in \varphi - PA_Q(c)$. Furthermore, from $Q \overset{d}{\not\rightarrow}$, we have $Q_j \overset{d}{\not\rightarrow}$ for some $j \in PA_Q(d)$. We have three subcases.

Subcase 2.1: $i \in PA_Q(d)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(d)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$), the left-sequence introduction transformation definition (15), $P_j \xrightarrow{c}$, and $Q_j \overset{d}{\not\rightarrow}$, we have $Q_j = P_j[c/d;c,nl]$. Hence $Q \neq P[c/d;c]$. Thus $Q = P[c/d;c,nl]$. We therefore conclude $Q_i = P_i[c/d;c,nl]$ (for all $i \in PA_Q(d)$).

Since $P_i \xrightarrow{c}$ and $i \in PA_Q(d)$, $body(P_i)$ must have the form $(c;F);*E$ as discussed above (page 16). By the transition relation definition (7), rule **Act, Seq**, we have

$((c;F);*E) \xrightarrow{c} (F;*E)$

Hence we have

$body(P'_i) = (F; *E)$.

Now $Q_i = P_i[c/d; c, nl]$. Since $body(P_i) = (c; F); *E$, we have $body(Q_i) = (c; F_t); *E_t$, by the left-sequence introduction transformation (15). By the transition relation definition (7), rule **Act**, we have

$(c; F_t; *E_t) \xrightarrow{c} (F_t; *E_t)$

Since $body(Q_i) = ((c; F_t); *E_t)$, there exists a Q'_i such that $Q_i \xrightarrow{c} Q'_i$, and $body(Q'_i) = (F_t; *E_t)$. Since $body(P'_i) = (F; *E)$, we have $Q'_i = P'_i[c/d; c]$ by the left-sequence introduction transformation (15).

Subcase 2.2: $i \in PA_Q(c) - PA_Q(d)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in PA_Q(d)$. By ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Letting Q'_i be P'_i, we get $Q_i \xrightarrow{c} Q'_i$ and $Q'_i = P'_i$.

Subcase 2.3: $i \in \varphi - PA_Q(c)$.

Consider P_i, Q_i, P'_i for an arbitrary $i \in \varphi - PA_Q(c)$. By ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Since $P_i = P'_i$ in this case, we let $Q'_i = Q_i$, and hence $Q'_i = P'_i$.

If we now consider $Q' = (\| \ i \in \varphi : Q'_i)$, where the Q'_i are as given by the preceding case analysis, we see that $Q \xrightarrow{c} Q'$ by the transition relation definition (7), and also that $Q' = P'[c/d; c]$, by the left-sequence introduction transformation definition (15), since the above three cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P, Q) \mid Q \in P[c/d; c] \text{ or } Q \in P[c/d; c, nl]\}$, we have $P' \, \mathcal{S} \, Q'$ as required.

Clause 4: for all $a \in alphabet(P) - \{c\}$, if $Q \xrightarrow{a} Q'$, then $P \xrightarrow{a} P'$ for some P' such that $P' \, \mathcal{S} \, Q'$

Let $Q \xrightarrow{a} Q'$ for some $Q' = (\| \ i \in \varphi : Q'_i)$ and $a \in alphabet(P) - \{c\}$. We show that there exists a $P' = (\| \ i \in \varphi : P'_i)$ such that $P \xrightarrow{a} P'$ and $P' \, \mathcal{S} \, Q'$. By $Q \xrightarrow{a} Q'$ and the transition relation definition (7), $Q_i \xrightarrow{a} Q'_i$ for all $i \in PA_Q(a)$, and $Q_i = Q'_i$ for all $i \in \varphi - PA_Q(a)$. We have two cases.

Case 1: $i \in PA_Q(a)$.

Consider P_i, Q_i, Q'_i for an arbitrary $i \in PA_Q(a)$. We have two subcases, $i \in PA_Q(a) \cap PA_Q(d)$ and $i \in PA_Q(a) - PA_Q(d)$.

Subcase 1.1: $i \in PA_Q(a) \cap PA_Q(d)$.

From ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$), the left-sequence introduction transformation definition (15), and the subcase condition $i \in PA_Q(a) \cap PA_Q(d)$, we have ($Q_i = P_i[c/d; c]$ or $Q_i = P_i[c/d; c, nl]$). From $Q_i \xrightarrow{a}$, we get $Q_i \overset{c}{\not\to}$, since no occurrence of c in $Q_i, i \in PA_Q(d)$ is an operand of $[\!]$. Hence $Q_i \neq P_i[c/d; c, nl]$. Thus $Q_i = P_i[c/d; c]$. Since $Q_i \xrightarrow{a}$, we have $body(Q_i) = (a; F' \ [\!] \ G'); *E'$ for some action expressions F', G', E', as discussed above (page 16). From $Q_i = P_i[c/d; c]$ and the left-sequence introduction transformation definition (15), we conclude that $body(P_i) = (a; F \ [\!] \ G); *E$ where action expressions E, F, G are such that $F' = F_t, G' = G_t, E' = E_t$.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$((a; F_t \; [\!] \; G_t); *E_t) \overset{a}{\rightarrow} (F_t; *E_t)$

In general, Q_i can have more than one a-derivative, since G_t itself could have the form $a; F'_t \; [\!] \; G'_t$. However, there is no loss of generality in assuming that the execution of the action a in $a; F_t$ leads to Q'_i. Hence we have $body(Q'_i) = (F_t; *E_t)$. Now $Q_i = P_i[c/d; c]$. Since $body(Q_i) = (a; F_t \; [\!] \; G_t); *E_t$, we have $body(P_i) = (a; F \; [\!] \; G); *E$, by the left-sequence introduction transformation (15). By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$((a; F \; [\!] \; G); *E) \overset{a}{\rightarrow} (F; *E)$

Since $body(P_i) = (a; F \; [\!] \; G); *E$, there exists a P'_i such that $P_i \overset{a}{\rightarrow} P'_i$ and $body(P'_i) = F; *E$. Since $body(Q'_i) = F_t; *E_t$, we have $Q'_i = P'_i[c/d; c]$ by the left-sequence introduction transformation (15).

Subcase 1.2: $i \in PA_Q(a) - PA_Q(d)$.

From ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have $Q_i = P_i$. Letting P'_i be Q'_i we get $P_i \overset{a}{\rightarrow} P'_i$ and $P'_i = Q'_i$.

Case 2: $i \in \varphi - PA_Q(a)$.

Consider P_i, Q_i, Q'_i for an arbitrary $i \in \varphi - PA_Q(a)$. We have two subcases, $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$ and $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

Subcase 2.1: $i \in (\varphi - PA_Q(a)) \cap PA_Q(d)$.

From ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have ($Q_i = P_i[c/d; c]$ or $Q_i = P_i[c/d; c, nl]$). Since $i \in \varphi - PA_Q(a)$ these Q_i do not participate in action a. Therefore we have $Q'_i = Q_i$. Let $P'_i = P_i$. Since ($Q_i = P_i[c/d; c]$ or $Q_i = P_i[c/d; c, nl]$), we have ($Q'_i = P'_i[c/d; c]$ or $Q'_i = P'_i[c/d; c, nl]$).

Subcase 2.2: $i \in (\varphi - PA_Q(a)) - PA_Q(d)$.

From ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$), the left-sequence introduction transformation definition (15), and the subcase condition, we have $Q_i = P_i$. Since $i \in \varphi - PA_Q(a)$ these Q_i do not participate in action a. Letting P'_i be Q'_i, we have $Q_i = P_i = Q'_i = P'_i$.

Now suppose $PA_Q(a) \cap PA_Q(d) \neq \emptyset$. Then, by subcase 1.1, $Q'_i = P'_i[c/d; c]$ for some $i \in PA_Q(a) \cap PA_Q(d)$. Thus, by all the subcases above and definition 15, we conclude $Q' = P'[c/d; c]$. On the other hand, if $PA_Q(a) \cap PA_Q(d) = \emptyset$, then $(\varphi - PA_Q(a)) \cap PA_Q(d) = PA_Q(d)$, and therefore, by subcases 1.2, 2.1, 2.2, and definition 15, we conclude ($Q' = P'[c/d; c]$ or $Q' = P'[c/d; c, nl]$). Since $\mathcal{S} = \{(P, Q) \mid Q = P[c/d; c] \text{ or } Q = P[c/d; c, nl]\}$, we have $P' \, \mathcal{S} \, Q'$ in all cases.

If we now consider $P' = (\| \, i \in \varphi : P'_i)$, where the P'_i are as given by the preceding case analysis, we see that $P \overset{a}{\rightarrow} P'$ by the transition relation definition (7). Thus, clause 4 is satisfied.

<u>Clause 5: if $Q \overset{d}{\rightarrow} Q'$, then $P \, \mathcal{S} \, Q'$, and if $Q \overset{c}{\rightarrow} Q'$, then $P \overset{c}{\rightarrow} P'$ for some P' such that $P' \, \mathcal{S} \, Q'$</u>

Let us start with the first conjunct, namely "if $Q \xrightarrow{d} Q'$, then $P \mathcal{S} Q'$."

Let $Q \xrightarrow{d} Q'$ for some $Q' = (\| \; i \in \varphi : Q'_i)$. We show that $P \mathcal{S} Q'$. By $Q \xrightarrow{d} Q'$ and the transition relation definition (7), $Q_i \xrightarrow{d} Q'_i$ for all $i \in PA_Q(d)$ and $Q_i = Q'_i$ for all $i \in \varphi - PA_Q(d)$. We have two cases.

Case 1: $i \in PA_Q(d)$

Consider Q_i, P_i, Q'_i for an arbitrary $i \in PA_Q(d)$. From ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$), the left-sequence introduction transformation definition (15), and the case condition $i \in PA_Q(d)$, we have ($Q_i = P_i[c/d; c]$ or $Q_i = P_i[c/d; c, nl]$). From $Q_i \xrightarrow{d}$, we have $Q_i \neq P_i[c/d; c, nl]$. Hence $Q_i = P_i[c/d; c]$. From this, $Q_i \xrightarrow{d}$, and $i \in PA_Q(d)$, we have $body(Q_i) = (d; c; F'); *E'$ for some action expressions F', E', as discussed above (page 16). From $Q_i = P_i[c/d; c]$ and the left-sequence introduction transformation definition (15), we conclude that $body(P_i) = (c; F); *E$ where action expressions E, F are such that $F' = F_t, E' = E_t$.

By the transition relation definition (7), rule **Act**, we have

$((d; c; F_t); *E_t) \xrightarrow{d} ((c; F_t); *E_t)$

Thus, we may assume, without loss of generality, that

$body(Q'_i) = ((c; F_t); *E_t)$.

Since $body(P_i) = (c; F); *E$, we have $Q'_i = P_i[c/d; c, nl]$ by the left-sequence introduction transformation (15).

Note that the assumption that no occurrence of c in $P_i, i \in PA_Q(d)$, is an operand of $[\!]$ is crucial in carrying through the proof in this case. Suppose that this assumption is dropped. Then the form of $body(Q)_i$ becomes $body(Q_i) = (d; c; F' \; [\!] \; G'); *E'$. Also, $body(P_i) = (c; F \; [\!] \; G); *E$. Now $((d; c; F_t \; [\!] \; G_t); *E_t) \xrightarrow{d} ((c; F_t); *E_t)$, and so $body(Q'_i) = ((c; F_t); *E_t)$. However, since $body(P_i) = (c; F \; [\!] \; G); *E$, we get $P_i[c/d; c, nl] = ((c; F_t \; [\!] \; G_t); *E_t)$, and so we cannot conclude $Q'_i = P_i[c/d; c, nl]$, due to the difference introduced by the presence of the action expression G in P, which can occur only if c is allowed to be the operand of $[\!]$.

Case 2: $i \in \varphi - PA_Q(d)$.

Consider P_i, Q_i, Q'_i for an arbitrary $i \in \varphi - PA_Q(d)$. By ($Q = P[c/d; c]$ or $Q = P[c/d; c, nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Since $Q_i = Q'_i$ in this case, we have $Q'_i = P_i$.

From the preceding case analysis, we see that $Q' = P[c/d; c, nl]$, by the left-sequence introduction transformation definition (15), since the above two cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P, Q) \mid Q = P[c/d; c]$ or $Q = P[c/d; c, nl]\}$, we have $P \mathcal{S} Q'$ as required. Thus the first conjunct is satisfied.

(end of proof of first conjunct)

We next establish the second conjunct of the clause, namely "if $Q \xrightarrow{c} Q'$, then $P \xrightarrow{c} P'$ for some P' such that $P' \mathcal{S} Q'$."

Let $Q \xrightarrow{c} Q'$ for some $Q' = (\| \; i \in \varphi : Q'_i)$. We show that there exists a $P' = (\| \; i \in \varphi : P'_i)$ such that $P \xrightarrow{c} P'$ and $P' \mathcal{S} Q'$. By $Q \xrightarrow{c} Q'$ and the transition relation definition (7), $Q_i \xrightarrow{c} Q'_i$ for all $i \in PA_Q(d)$, $Q_i \xrightarrow{c} Q'_i$ for all

$i \in PA_Q(c) - PA_Q(d)$, and $Q_i = Q'_i$ for all $i \in \varphi - PA_Q(c)$. We have three cases.

Case 1: $i \in PA_Q(d)$

From ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) the left-sequence introduction transformation definition (15), and $i \in PA_Q(d)$, we have ($Q_i = P_i[c/d;c]$ or $Q_i = P_i[c/d;c,nl]$). From $Q_i \xrightarrow{c} Q'_i$, we have $Q_i \neq P_i[c/d;c]$. Hence $Q_i = P_i[c/d;c,nl]$. Since $Q_i \xrightarrow{c}$, we have $\mathit{body}(Q_i) = (c;F' \parallel G');{*E'}$ for some action expressions F', G', E', as discussed above (page 16). (Note that we can actually show $\mathit{body}(Q_i) = (c;F');{*E'}$ by using the $i \in PA_Q(d)$ and our assumption that c does not occur in a choice in $P_i, i \in PA_Q(d)$. However, this is not needed here, and so we can use the more general form $\mathit{body}(Q_i) = (c;F' \parallel G');{*E'}$. In reality, we will always have $G' = 0$.) Thus $\mathit{body}(P_i) = (c;F \parallel G);{*E}$ where action expressions E, F, G are such that $F' = F_t, G' = G_t, E' = E_t$.

By the transition relation definition (7), rules **Act, Ch, Seq** applied in sequence, we have

$$((c;F_t \parallel G_t);{*E_t}) \xrightarrow{c} (F_t;{*E_t})$$

In general, Q_i can have more than one c-derivative, since G_t itself could have the form $c;F'_t \parallel G'_t$. However, there is no loss of generality in assuming that the execution of the action c in $c;F_t$ leads to Q'_i. Hence we have $\mathit{body}(Q'_i) = (F_t;{*E_t})$. Now $\mathit{body}(P_i) = (c;F \parallel G);{*E}$. By the transition relation definition (7), rules **Act, Ch, Seq**, we have

$$((c;F \parallel G);{*E}) \xrightarrow{c} (F;{*E})$$

Since $\mathit{body}(P_i) = (c;F \parallel G);{*E}$, there exists a P'_i such that $P_i \xrightarrow{c} P'_i$ and $\mathit{body}(P'_i) = F;{*E}$. Since $\mathit{body}(Q'_i) = F_t;{*E_t}$, we have $Q'_i = P'_i[c/d;c]$ by the left-sequence introduction transformation (15).

Case 2: $i \in PA_Q(c) - PA_Q(d)$.

Consider P_i, Q_i, Q'_i for an arbitrary $i \in PA_Q(c) - PA_Q(d)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Letting P'_i be Q'_i, we get $P_i \xrightarrow{c} P'_i$ and $Q'_i = P'_i$.

Case 3: $i \in \varphi - PA_Q(c)$.

Consider Q_i, P_i, Q'_i for an arbitrary $i \in \varphi - PA_Q(c)$. By ($Q = P[c/d;c]$ or $Q = P[c/d;c,nl]$) and the left-sequence introduction transformation definition (15), we have $Q_i = P_i$. Now $Q_i = Q'_i$ in this case as explained above. Letting $P'_i = P_i$ we get $P'_i = Q'_i$.

If we now consider $P' = (\| \; i \in \varphi : P'_i)$, where the P'_i are as given by the preceding case analysis, we see that $P \xrightarrow{c} P'$ by the transition relation definition (7), and also that $Q' = P'[c/d;c]$, by the left-sequence introduction transformation definition (15), since the above three cases cover the entire indexing set φ. Since $\mathcal{S} = \{(P,Q) \mid Q = P[c/d;c] \text{ or } Q = P[c/d;c,nl]\}$, we have $P' \, \mathcal{S} \, Q'$ as required.

We have shown that all five clauses of definition 16 hold, hence $\mathcal{S}$ is a *dc*-bisimulation. □

Proposition 9 *Let $Q = P[c/d;c]$. If Q' is an arbitrary derivative of Q, then there exists a derivative P' of P such that $P' \sim Q'$.*

Proof: Let $\mathcal{S} = \{(P,Q) \mid Q = P[c/d;c]\}$. By proposition 8, $\mathcal{S}$ is a dc-bisimulation. Hence $\mathcal{S} \subseteq \sim$ by definition 17. Since $P\,\mathcal{S}\,Q$ by assumption, we conclude $P \sim Q$. We now establish the proposition by induction on the number of steps of derivation of program Q'. We let Q^i represent a derivative of Q which has been obtained after i events. Thus Q', the derivative of Q after n steps, is represented by Q^n.

Base Case.

We first prove the base case for one step of the derivation. Let $Q \xrightarrow{a_1} Q^1$. There are three cases, depending on a_1. First, if $a_1 = d$, from $P \sim Q$ and proposition 7 we conclude that $P \sim Q^1$. Second, if $a_1 = c$, from $P \sim Q$ and proposition 7 we conclude that there exists P^1, a derivative of P ($P \xrightarrow{c} P^1$) such that $P^1 \sim Q^1$. Third, if $a_1 \neq c$ and $a_1 \neq d$, then from $P \sim Q$ and proposition 7 we conclude that there exists P^1, a derivative of P ($P \xrightarrow{a_1} P^1$) such that $P^1 \sim Q^1$. Thus the base case is established.

Induction Hypothesis. Let us assume that our proposition holds up to n steps of the derivation.

Induction Step. Let $Q \xrightarrow{a_1} \cdots \xrightarrow{a_n} Q^n$ for some path $\rho = a_1, \ldots, a_n$ of actions. By our induction hypothesis, there exists a derivative P' of P such that $P' \sim Q^n$. Let $Q^n \xrightarrow{a_{n+1}} Q^{n+1}$. Using the same argument as the base case, we conclude that there exists a derivative P'' of P' such that $P'' \sim Q^{n+1}$. Since P'' is a derivative of P' which is a derivative of P, P'' is a derivative of P. Thus our induction step is established. □

As before we prove that our syntactic transformation preserves the property of deadlock-freedom.

Theorem 10 *Let $Q = P[c/d;c]$. If P is deadlock-free, then so is Q.*

Proof: Let Q' be an arbitrary derivative of Q. By definition 14, it suffices to show $Q' \xrightarrow{a}$ for some action a. By proposition 9, we have

If Q' is an arbitrary derivative of Q, then there exists a derivative P' of P such that $P' \sim Q'$

By assumption, P is deadlock-free. Hence, by definition 14, $P' \xrightarrow{a'}$ for some a'. If $a' \neq c$, then $Q' \xrightarrow{a'}$ by proposition 7. If $a' = c$, then either $Q' \xrightarrow{dc}$, or $Q' \xrightarrow{c}$ again by proposition 7. Hence in all cases we have $Q' \xrightarrow{a}$ for some a, and therefore Q is deadlock-free. □

5 Example: The Elevator Problem

We now apply the transformations to a specification of the elevator control problem [Davis 87]. Our version of this problem is defined as follows: An N elevator system is to be installed in a building with F floors. The internal mechanisms of the elevators are given. The problem is to design the logic control that moves elevators between floors according to the following constraints:

- Each elevator has a set of buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The button illumination is canceled when the elevator stops at that floor.
- Each floor (except ground and top) has two buttons: one to request an up-elevator and one to request a down-elevator. These buttons also illuminate when pressed. A floor button is canceled when an elevator traveling in the desired direction visits the floor.
- All requests for elevators (from floors and within elevators) must be served within a finite amount of time.

Figure 1 gives an initial solution to this problem. The actions in figure 1 perform the following functions:

select Indicates that a particular floor button has been pressed, selects a particular elevator to service the floor button request, and enters that request into the schedule of the elevator.

p–schedule Indicates that a particular panel button has been pressed and enters the panel button request into the schedule of the elevator containing the panel button.

f–satisfy Indicates that a floor button request has been satisfied; i.e., the elevator has visited the desired floor and was traveling in the desired direction.

p–satisfy Indicates that a panel button request has been satisfied, i.e., the elevator has visited the desired floor.

The *select* and *p–schedule* actions are very large-grain — they each represent several activities. We shall now transform each of these actions into a sequence of actions. *select* models the sequence: press floor button; select an elevator to service the floor button request; enter that request into the schedule of the elevator. We wish to model each of these activities by its own action. It seems appropriate that, in the decomposed sequence of actions, *select* will model the middle activity, i.e., select an elevator to service the floor button request. This then requires that we apply a left-sequence introduction transformation with $d = f\text{–}press$ and $c = select$. The *f–press* action thus introduced models the pressing of the floor button. Next, we apply a right-sequence introduction transformation with $c = select$ and $d = f\text{–}insert$. The *f–insert* action thus introduced models entering the request into the elevator's schedule. Hence, the granularity of *select* at the modeling level has been reduced since it now models only the activity of selecting an elevator to service the floor button request, rather then the sequence: press floor button; select elevator; enter request in elevator's schedule. In an analogous manner, we refine the *p–schedule* action into the sequence *p–press*; *p–schedule*; *p–insert*. The resulting program is shown in figure 2.

In figure 2, we still consider the *select* action to have large granularity, since selection of an elevator to service the floor button request requires three rounds of communication between the floor button and the various elevator controllers (broadcast from button to controllers, reply from controllers to button, and broadcast from button to controllers). We now refine the action *select* to the sequence *select*; *reply*; *choose*; *inform*. The actions *select*; *reply*; *inform* each model only a single round of communication, while the action *choose* is a local action. This refinement is achieved by four applications of the right-sequence introduction transformation (in all of which we have $c = select$). The final result is shown in figure 3. Note that we have changed the name of *select* to *select–announce*, since it now only models the announcement of the floor button request to all elevators. Also, the actions *f–satisfy* and *p–satisfy* have not been subject to any transformations, and so retain the functionality they had in figure 1 (given in the list above).

Overall, the actions that have been introduced into the program of figure 1 have the following functions:

reply The reply from the elevators to the announcement of the floor button request. This reply contains information (i.e., the elevator's location and direction of travel) that the floor button (*f–button*) process uses to select a particular elevator to service its request.

choose The (local) action of the floor button process in which it selects the elevator to service its request.

inform The action in which the floor button process informs all the elevators of its choice.

f–press The (local) action that indicates that a floor button has been pressed.

p–press The (local) action that indicates that a panel button has been pressed.

f–insert The (local) action in which the chosen elevator adds the floor button request to its schedule.

p–insert The (local) action in which the elevator adds the panel button request to its schedule.

Although it is quite easy to verify by inspection that the program of figure 1 is deadlock-free, the same property is not so readily apparent for the program of figure 3. However, we know that the latter program *is* deadlock-free, because it was derived from the former using only transformations that preserve deadlock-freedom. Of course, some additional effort would enable us to verify the deadlock-freedom of the program of figure 3, since this example is quite simple. With a program of realistic size, however, the difference in size between the initial and final programs would often be substantial. Hence, the time complexity of automatic verification of deadlock freedom would be significantly smaller for the initial program.

6 Conclusions and Future Work

In this paper we have described two syntactic transformations for program refinement. They are used for introducing new actions in sequence with pre-existing actions and may be viewed as tools for decomposing a large action into a sequence of smaller actions. Such decomposition is a natural step in the process of refining programs. We proved formally that our transformations preserve deadlock-freedom. Our next step is to identify the safety and liveness properties that are preserved by these transformations. For example, there is good reason to believe

```
 f–button :: ∗[ select ; f–satisfy ]
||
 p–button :: ∗[ p–schedule ; p–satisfy ]
||
 elevator–controller :: *[ select
                            ▯
                           p–schedule
                            ▯
                           f–satisfy
                            ▯
                           p–satisfy
                           ]
```

Figure 1: Zero-Level Elevator System

```
 f–button :: ∗[ f–press ; select ; f–satisfy ]
||
 p–button :: ∗[ p–press ; p–schedule ; p–satisfy ]
||
 elevator–controller :: *[ select ; f–insert
                            ▯
                           p–schedule ; p–insert
                            ▯
                           f–satisfy
                            ▯
                           p–satisfy
                           ]
```

Figure 2: First-Level Elevator System

```
 f–button :: ∗[ f–press ; select–announce ; reply ; choose ; inform ; f–satisfy ]
||
 p–button :: ∗[ p–press ; p–schedule ; p–satisfy ]
||
 elevator–controller :: *[ select–announce ; reply ; inform ; f–insert
                            ▯
                           p–schedule ; p–insert
                            ▯
                           f–satisfy
                            ▯
                           p–satisfy
                           ]
```

Figure 3: Second-Level Elevator System

that the transformations preserve the safety property of *mutual exclusion* — if action c occurs inside a critical region, then it will still be inside the critical region in the transformed program. Additionally, one could include action d in the critical region if the transformed program results from applying the right-sequence introduction transformation (but not if it results from applying the left-sequence introduction transformation).

We note that the formal correctness proofs for the transformations are lengthy, especially since the correctness of the transformations themselves is somewhat apparent. The salient point, however, is that the proofs are, in effect, reused each time the transformations are applied. A more traditional proof rule for program correctness [Chandy et. al. 88, Francez 92, Lamport 80] has a shorter formal justification, but requires the designer to produce a manual proof each time the rule is applied. We believe that it is much more efficient to verify the correctness of a transformation that can be reused many times, even if the proof is somewhat lengthy.

Correctness-preserving transformations for distributed systems are, in principle, a foundation for the eventual goal of compiling abstract specifications into architecturally adequate code. Those who find that objective too distant should, nevertheless, be interested in the medium-term goal of automating certain laborious and error-prone parts of the development process. An interactive compiler that handles much of the labor — and is guaranteed not to introduce the deadlocks and other errors that plague concurrent systems — would be valuable, even if it still depends heavily on human design creativity. This research is designed to support both the medium- and the long-term goals.

Acknowledgments. We would like to thank Mike Evangelist and Ira Forman for extensive discussions on the nature and application of program transformations.

References

[Aceto 92] ACETO, L., 1992, Action Refinement in Process Algebras. D. Phil. thesis, University of Sussex, (Cambridge University Press).

[Back et. al. 83] BACK, R.J.R., and KURKI-SUONIO, R., 1989, Decentralization of Process Nets With Centralized Control. *Distributed Computing* 3: 73–87, (1989).

[Back et. al. 85] BACK, R.J.R., and KURKI-SUONIO, R., 1985, Serializability in Distributed Systems With Handshaking. Carnegie Mellon University TR 85-109.

[Baeten et. al. 90] BAETEN, J. C., and WEIJLAND, P., 1990, Process Algebra. Cambridge Tracts in Theoretical Computer Science.

[Chandy et. al. 88] CHANDY, K.M., and MISRA, J., 1988, Parallel Program Design. Addison-Wesley.

[Cleaveland et. al. 96] CLEAVELAND, R., and PANANGADEN, P., Type Theory and Concurrency. *International Journal of Parallel Programming* 17(2): 153 – 206, (1988).

[Constable et. al. 89] CONSTABLE, R., and HOWE, D., Nuprl as a General Logic. Technical Report 89-1021, Department of Computer Science, Cornell University.

[Czaja et. al. 91] CZAJA, I., VAN GLABEEK, R., and GOLTZ, U., 1991, Interleaving Semantics and Action Refinement with Atomic Choice. Arbeitspapiere der GMD 594, Sankt Augustin.

[Davis 87] DAVIS, N., 1987, Proceedings of the Fourth International Workshop on Software Specification and Design. IEEE Computer Society Press.

[Francez 92] FRANCEZ, N., 1992, Program Verification. Addison-Wesley.

[van Glabeek 90] VAN GLABEEK, R., 1990, Comparative Concurrency Semantics and Refinement of Actions. Ph.D. dissertation, Vrije Univerity of Amsterdam.

[Hoare 69] HOARE, C.A.R. Hoare, 1969, An Axiomatic Basis for Computer Programming. *CACM* 12 (10): 576–580.

[Hoare 78] HOARE, C.A.R., 1978, Communicating Sequential Processes. *CACM* 21 (8): 666–678.

[Hoare 85] HOARE, C.A.R., 1985, Communicating Sequential Processes. Prentice-Hall.

[Lamport 80] LAMPORT, L., 1980, The 'Hoare Logic' of Concurrent Programs. *Acta Informatica*, 14: 21–37.

[Manna et. al. 94] MANNA, Z., ANUCHITANUKUL, A., BJORNER, N., BROWNE, A., CHANGG, E., COLON, M., de ALFARO, L., DEVARJAN, H., SIPMA, H., and UNRIBE, T, STeP: the Stanford temporal prover. Technical Report, Dept. of Computer Science, Stanford University.

[Milner 89] MILNER, R., 1989, Communication and Concurrency. Prentice-Hall.

[Ramesh et. al. 87] RAMESH, S., and MEHNDIRATTA, H., 1987, A methodology for developing distributed programs. *IEEE Transactions on Software Engineering* SE-13 (8): 967–976.

[Vardi 87] VARDI, M.Y., 1987, Verification of Concurrent Programs, The Automata Theoretic Framework. *Logic in Computer Science.*