Simple Section Interchange and Properties of Non-Computable Functions

Ward Douglas Maurer

Computer Science Department, George Washington University, Washington, DC 20052, USA

Abstract

We here extend our earlier work on the theory of partially defined computer instructions and guards to cover partially defined computer expressions and programs. The notion of the relevant region of an expression is generalized to conditional relevant regions, and we specify upper bounds on these for expressions built up from simpler ones. We then proceed to specify upper bounds on the input and output regions of programs containing goto statements, even though the state transformations of such programs are not necessarily computable. This is then combined with our earlier commutativity results to obtain a general condition under which two simple sections of such a program commute with each other, and therefore may be interchanged, or possibly done in parallel.

Key words: instruction theory, non-computable functions, formal methods, partially defined instructions

1 Overview

We define a simple section of a program to be a subset of its statements, having a single entry and a single exit. There is no restriction on jumps or loops within a simple section, as there is with basic blocks [1]. It follows that simple sections of a program are often quite numerous; indeed, a program with \( n \) phases, always executed in sequence, has a simple section consisting of phases \( i \) through \( j \), for any \( i \) and \( j \), \( 1 \leq i \leq j \leq n \). We here take up the question of when two adjacent simple sections of a program \( P \) can be

---

Email address: maurer@gwu.edu (Ward Douglas Maurer).
interchanged, within $P$, without changing the effect of $P$; and this will be true if their overall effects, or state transformations, commute with each other.

Our main result, given as Theorem 6.1 below, extends our earlier work on interchanging two adjacent parts of a program. The simplest case, for this problem, is that in which these are individual instructions on a computer. In [8] we have given a criterion for this, in terms of the input and output regions of their effects, when these are everywhere defined. In [9] we removed that restriction. If $\Sigma$ and $\Sigma'$ are the parts, and $e_\Sigma$ and $e_{\Sigma'}$ are their effects, then $e_\Sigma(S)$ is only guaranteed to be defined when $e_{\Sigma}(S) = true$, where $e_{\Sigma}: \mathbb{S} \rightarrow \{true, false\}$ is a function which we call the guard of $e_\Sigma$, and similarly for $e_{\Sigma'}$ and $e_{\Sigma'}$. We then defined the conditional input and output region of an instruction with a guard, and extended many of the results of [8] to the conditional case, including our commutativity criterion.

The main result of this paper holds even when we do not know what $\Sigma$ and $\Sigma'$ do, or indeed whether they are computable. All that is needed is for $\Sigma$, $\Sigma'$, and the union of $\Sigma$ and $\Sigma'$, all to be simple sections of a program, whose conditional regions satisfy the criteria required in [9]. This is rigorously proved, and it is therefore necessary to provide mathematical definitions of the various parts of a program, all of which are (possibly) partially defined. These include programs, statements, and left- and right-side expressions; their associated functions of the state; and their conditional input, output, and relevant regions.

A program $P$, as well as each statement of $P$, has an effect which has a conditional input and output region, as these were defined in [9]. Each statement of $P$ also has a controller, specifying the next statement when given the current state. These controllers, as well as the expressions which define a statement, have conditional relevant regions. We define these here, and show their basic properties; they are extensions of the concept of relevant region defined in [9]. In Theorem 5.1 below, we show that the conditional output region of the effect of $P$ is contained in the union of the conditional output regions of the effects of its statements. In Theorem 5.2, we show that the conditional input region of the effect of $P$ is contained in the union of the conditional input and output regions of the effects of its statements and the conditional relevant regions of its controllers. Again, these results hold even if the effect of $P$ is non-computable.

Several basic constructions and their properties are needed here. Given $P$ and $\Sigma$, we may form a new sequential program $P'$ which is like $P$ except that $\Sigma$ is contracted into a single statement of $P'$. We prove that this process does not change the effect of $P$, or its guard. The inverse of a contraction $P'$ of $P$ is an expansion $P$ of $P'$, which likewise preserves the effect of $P'$ and its guard. We prove that contraction and expansion are inverses of each other. A section
\( \Sigma' \) of \( P \) may be embedded in another section \( \Sigma \) of \( P \); here \( \Sigma' \) has the same properties as a section of \( \Sigma \) that it has as a section of \( P \). When a section \( \Sigma \) is contracted, a second section \( \Sigma' \) is transformed into an induced section of the contraction, which preserves the properties of \( \Sigma' \). There is also the general concept of a relocation of \( \Sigma \) to a new start index; and this also preserves the properties of \( \Sigma \). All this is then used in the proof of Theorem 6.1.

2 Summary of Previous Results

We use the following definitions, theorems, etc., from [9].

**Axiom 1** (of [9]) If \( S_1, S_2 \in \mathcal{S} \), and \( M' \) is any subset of \( M \), then the state \( S_3 \), with

\[
S_3(x) = \begin{cases} 
S_1(x) & x \in M' \\
S_2(x) & x \notin M'
\end{cases}
\]

is a member of \( \mathcal{S} \).

**Axiom 2** (of [9]) If \( S_1, S_2 \in \mathcal{S} \), then \( \{ x \in M : S_1(x) \neq S_2(x) \} \) is finite.

**Definition 2.1** (Definition 2 of [9]) Let \( (M_1, B_1, S_1, I_1) \) and \( (M_2, B_2, S_2, I_2) \) be two computers. A function \( I : S_1 \rightarrow S_2 \) is a domain operation on \( S_1 \).

**Definition 2.2** (Definition 3 of [9]) Given a domain operation \( I \) as in Definition 2.1, its relevant region \( \text{RR}(I) \) is defined by \( \text{RR}(I) = \{ x \in M_1 : \text{there exist } S_1 \text{ and } S_2 \in S_1 \text{ such that } S_1(z) = S_2(z) \text{ for all } z \in M_1 \neq x, \text{ but } I(S_1) \neq I(S_2) \} \).

**Definition 2.3** (Definition 4 of [9]) Let \( (M, S, I) \) be a computer as in Definition 4.3 of [8], where \( S \) is a cartesian product of the various \( B_x \). Let \( M' \) be a subset of \( M \), and let \( S' \) be the cartesian product of only those \( B_x \) for \( x \in M' \); then \( \text{Sub}(S, M') = S' \) is a subdomain of \( S \). Equivalently, let \( (M, B, S, I) \) be a computer as in Definition 3.1 of [8], where \( S \) is a set of maps \( S : M \rightarrow B \) satisfying Axioms 1 and 2, and let \( S' \) be the set of all maps of the form \( S|_{M'} \), where \( S \in S \). Then \( \text{Sub}(S, M') = S' \) is a subdomain of \( S \).

**Definition 2.4** (Definition 5 of [9]) A computer expression, or C-expression, with type-set \( T \), on the domain \( \mathcal{S} \) of a computer \( (M, B, S, I) \) or \( (M, S, I) \) is a function \( f : \mathcal{S} \rightarrow T \). The relevant region of \( f \) is the set \( \{ x \in M : \text{there exist } S_1 \text{ and } S_2 \in \mathcal{S} \text{ such that } S_1(z) = S_2(z) \text{ for all } z \in M \neq x, \text{ but } f(S_1) \neq f(S_2) \} \).
Here $T$ is often, although not always, contained in the $B$ of $(M, B, S, \mathbb{I})$ or $(M, S, \mathbb{I})$. For example, $B$ might be $\{0, 1\}$, on a computer regarded as entirely binary, with each state being a vector of current values of individual bits, whereas $T$ might be the integers. Alternatively, $B$ might be finite, and contain the one-word integers, whereas $T$ is infinite, and consists of all the integers.

**Definition 2.5** (Definition 6 of [9]) A **Boolean C-expression** is a C-expression whose type-set is \{true, false\}.

**Theorem 2.1** (Theorem 4 of [9]) Let $(M, B, S, \mathbb{I})$ be a computer, let $T$ be a set, and let $f : S \rightarrow T$ be a C-expression. For any $S_1, S_2 \in S$, if $S_1 | RR(f) = S_2 | RR(f)$, then $f(S_1) = f(S_2)$.

**Definition 2.6** (Definition 8 of [9]) Let the domain operation $I : S_1 \rightarrow S_2$ be defined only on some subset $S'_1$ of $S_1$. A **guard** of $I$ is a Boolean C-expression on $S_1$, denoted by $I^g$, such that $S \in S'_1$ whenever $I^g(S) = true$. If $I^g(S) \equiv true$ for all $S \in S_1$, we sometimes say that $I$ has no guard.

**Definition 2.7** (Definition 11 of [9]) Let $(M, S, \mathbb{I})$ be a computer, where $B_z$, as usual, is the set of allowable values of $z$, for any $z \in M$. Then:

(a) If $k \in B_z$, for any $z \in M$, then $k$ is the C-expression $f : S \rightarrow \{k\}$ such that $f(S) = k$ for all $S \in S$.

(b) If $v \in M$, then $v$ is the C-expression $f : S \rightarrow B_v$ such that $f(S) = S(v)$ for all $S \in S$.

(c) The C-expressions $f : S \rightarrow \{true, false\}$ with $f(S) \equiv true$ and $f(S) \equiv false$ are denoted by true and false respectively.

**Theorem 2.2** (Theorem 5 of [9]) Using the notation of Definitions 2.4 and 2.7:

(a) $RR(k) = \emptyset$ and $RR(v) \subseteq \{v\}$.

(b) If $B_v$ has at least two elements, then $RR(v) = \{v\}$.

(c) $RR(true) = \emptyset$ and $RR(false) = \emptyset$.

(d) If $RR(I^g) = \emptyset$, then either $I^g = true$ or $I^g = false$.

**Definition 2.8** (Definition 12 of [9]) An **integer variable** of a computer $(M, S, \mathbb{I})$ is an element $z \in M$ such that $B_z$ is some set of integers, containing at least 0 and 1. A **constant integer** is an element $k \in B_z$, for some such $B_z$. The **six relations** on integers are $=, \neq, <, >, \leq$, and $\geq$. In this context:

(a) If $x$ is an integer variable, $k$ is a constant integer, and $R$ is one of the six relations, then $x R k$ is the C-expression $f : S \rightarrow \{true, false\}$ with $f(S) = true$ when $S(x) R k$ and $f(S) = false$ otherwise.

(b) If $x$ and $y$ are integer variables and $R$ is one of the six relations, then $x R y$ is the C-expression $f : S \rightarrow \{true, false\}$ with $f(S) = true$ when
\[ S(x) \text{ R } S(y) \text{ and } f(S) = \text{false otherwise}. \]

**Theorem 2.3** (Theorem 6 of [9]) Using the notation of Definition 2.8:

(a) If \( R = \text{or} \neq, \) and \( k \in B_x, \) then \( RR(x \ R k) = \{x\}. \)
(b) If \( x \neq y, \) then \( RR(x \ R y) = \{x, y\}. \)

**Definition 2.9** (Definition 15 of [9]) If \( f \) and \( g \) are Boolean C-expressions (as in Definition 2.5), then \( f \text{ and } g \) is the Boolean C-expression \( h \) such that \( h(S) = (f(S) \text{ and } g(S)) \) for all \( S \in \mathbb{S}. \)

**Theorem 2.4** (Theorem 9 of [9]) Using the notation of Definitions 2.4 and 2.9, we have:

(a) \( RR(h) \subseteq RR(f) \cup RR(g). \)
(b) If \( RR(f) \cap RR(g) = \emptyset, \) \( f \neq \text{false}, \) and \( g \neq \text{false}, \) then \( RR(h) = RR(f) \cup RR(g). \)

**Definition 2.10** (Definition 16 of [9]) Let \( (M, B, \mathbb{S}, \mathbb{I}) \) be a computer; let \( I: \mathbb{S} \to \mathbb{S} \) be defined only on a subset of \( \mathbb{S}; \) and let \( I^g: \mathbb{S} \to \{\text{true}, \text{false}\} \) be a Boolean C-expression, such that \( I(S) \) is defined whenever \( I^g(S) = \text{true}. \) Then the conditional output region \( \text{COR}(I, I^g) \subseteq M \) is defined as \( \text{COR}(I, I^g) = \{x \in M: \text{there exists } S \in \mathbb{S} \text{ such that } I^g(S) = \text{true and } S(x) \neq I(S)(x)\}. \)

**Definition 2.11** (Definition 17 of [9]) Let \( I_1: \mathbb{S} \to \mathbb{S} \) and \( I_2: \mathbb{S} \to \mathbb{S} \) have the respective guards \( I_1^g \) and \( I_2^g, \) as in Definition 2.6. Let \( J: \mathbb{S} \to \mathbb{S} \) be such that \( J(S) = I_2(I_1(S)) \), and let \( J^g \) be defined by \( J^g(S) = I_1^g(S) \& \& I_2^g(I_1(S)). \) Then \( J, \) with the guard \( J^g, \) is the composition of \( I_1 \) and \( I_2 \) with respective guards \( I_1^g \) and \( I_2^g \). We sometimes write \( (J, J^g) = (I_1, I_1^g) \circ (I_2, I_2^g); \) if \( I_1 \) and \( I_2 \) have no guards, we write \( J = I_1 \circ I_2. \)

**Definition 2.12** (Definition 18 of [9]) Under the conditions of Definition 2.10, the conditional input region \( \text{CIR}(I, I^g) \) is defined as \( \{x \in M: \text{there exist } S_1 \text{ and } S_2 \in \mathbb{S} \text{ such that:} \}

\begin{align*}
(a) & \quad S_1(z) = S_2(z) \text{ for all } z \neq x; \\
(b) & \quad I^g(S_1) = \text{true}; \text{ and} \\
(c) & \quad \text{either } I^g(S_2) = \text{false}, \text{ or else there exists } y \in \text{COR}(I, I^g) \text{ such that } I(S_1)(y) \neq I(S_2)(y)\}. 
\end{align*}

**Theorem 2.5** (Theorem 10 of [9]) Using the notation of Definition 2.12, for any states \( S_1, S_2 \in \mathbb{S}, \) if \( S_1 \text{CIR}(I, I^g) = S_2 \text{CIR}(I, I^g) \) and \( I^g(S_1) = I^g(S_2) = \text{true}, \) then \( I(S_1)(\text{COR}(I, I^g) = I(S_2)(\text{COR}(I, I^g). \) Also, \( RR(I^g) \subseteq \text{CIR}(I, I^g). \)

**Corollary 2.1** (Corollary 6 of [9]) Under the conditions of Definition 2.11, and if \( \text{COR}(I_1, I_1^g) \cap (\text{CIR}(I_2, I_2^g) \cup \text{COR}(I_2, I_2^g)) = \emptyset \) and \( \text{COR}(I_2, I_2^g) \cap
\[(CIR(I_1, I_1^g) \cup COR(I_1, I_1^g)) = \emptyset, \text{ we have } (I_1, I_1^g) \circ (I_2, I_2^g) = (I_2, I_2^g) \circ (I_1, I_1^g)\]

(that is, \(I_1\) and \(I_2\) commute).

### 2.1 Additional Preliminary Results

In [9] we introduced guards for instructions. We also introduced more general computer expressions, but we did not use these, except for guards, which are always defined. We did, however, give some examples of computer expressions which have guards (Examples 4, 6, and 7 in Section 4.1 of [9]). Here we take these up in greater generality. A computer expression, as presented in [9], is a special case of a domain operation \(I : S_1 \rightarrow S_2\), with \(S_2\) being a degenerate cartesian product of a single \(B_x = T\), and thus identified with \(T\) itself. We therefore specialize Definition 2.6 to this case.

**Definition 2.13** Let the computer expression \(f : S \rightarrow T\) be defined only on some subset \(S'\) of \(S\). A **guard** of \(f\) is a Boolean C-expression on \(S\), denoted by \(f^g\), such that \(S \in S'\) whenever \(f^g(S) = \text{true}\). If \(f^g(S) \equiv \text{true}\) for all \(S \in S\), we sometimes say that \(f\) has **no guard**.

It has been suggested to us that the phrase “no guard” is potentially confusing; it might sound as if the guard should be always false, rather than always true. Perhaps the best way to understand “no guard” is by analogy with a real guard. If a building has a guard, only certain (authorized) people can enter; if the building has no guard, anyone can enter. In the same way, if a function \(f(S)\) has a guard, it is defined only for certain states \(S\); if it has no guard, it is defined for any state \(S\).

An informal condition was given in [9], specifying when two instructions, with guards, are the same. Here we formalize this as a definition.

**Definition 2.14** (Last paragraph of Section 4.1 of [9]) Given the instructions \(I_1\) and \(I_2\), with the respective guards \(I_1^g\) and \(I_2^g\), the notation \((I_1, I_1^g) = (I_2, I_2^g)\) means that \(I_1^g = I_2^g\) and that \(I_1(S) = I_2(S)\) whenever \(I_1^g(S) = \text{true}\).

We shall also need a definition and a theorem which do for the logical or what Definition 2.9 and Theorem 2.4 do for the logical and (although note that the hypotheses of Theorem 2.6(b) are different from those of Theorem 2.4(b)).

**Definition 2.15** If \(f\) and \(g\) are Boolean C-expressions (as in Definition 2.5), then \(f\) **or** \(g\) is the Boolean C-expression \(h\) such that \(h(S) = (f(S) \text { or } g(S))\) for all \(S \in S\).

**Theorem 2.6** Using the notation of Definitions 2.4 and 2.15, we have:
(a) $RR(h) \subseteq RR(f) \cup RR(g)$.

(b) If $RR(f) \cap RR(g) = \emptyset$, $f \neq \text{true}$, and $g \neq \text{true}$, then $RR(h) = RR(f) \cup RR(g)$.

PROOF.

(a) Let $x \notin RR(f)$ and $x \notin RR(g)$. Then for any two states $S_1, S_2 \in \mathbb{S}$ such that $S_1(z) = S_2(z)$ for all $z \in M \neq x$, we have $f(S_1) = f(S_2)$ and $g(S_1) = g(S_2)$ by Definition 2.4. Therefore $h(S_1) = (f(S_1) \text{ or } g(S_1)) = (f(S_2) \text{ or } g(S_2)) = h(S_2)$, so that $x \notin RR(h)$ by Definition 2.4, and $RR(h) \subseteq RR(f) \cup RR(g)$.

(b) By part (a) above, it suffices to show that $RR(f) \cup RR(g) \subseteq RR(h)$. Let $x \in RR(f)$, so that there exist two states $S_1, S_2 \in \mathbb{S}$ such that $S_1(z) = S_2(z)$ for all $z \in M \neq x$, but $f(S_1) \neq f(S_2)$. Since $f(S_1)$ and $f(S_2)$ are Boolean and unequal, one of them (say $f(S_1)$) must be $\text{true}$, while $f(S_2)$ is false. Since $g \neq \text{true}$, there exists a state $S'$ such that $g(S') = \text{false}$. Let $S_3$ and $S_4$ be defined by $S_3(z) = S_1(z)$ for $z \in RR(f)$ (so that, by Theorem 2.1, we have $f(S_3) = f(S_1) = \text{true}$); $S_3(z) = S'(z)$ for $z \notin RR(f)$; $S_4(z) = S_2(z)$ for $z \in RR(f)$ (so that, by Theorem 2.1, we have $f(S_4) = f(S_2) = \text{false}$); and $S_4(z) = S'(z)$ for $z \notin RR(f)$. Here $S_3$ and $S_4 \in \mathbb{S}$ by Axiom 1. Since $RR(f) \cap RR(g) = \emptyset$, we have $S_3(z) = S_4(z) = S'(z)$ for all $z \in RR(g)$, so that $g(S_3) = g(S_4) = g(S')$ (by Theorem 2.1) = false. If $z \notin RR(f)$, then $S_3(z) = S'(z) = S_4(z)$; if $z \in RR(f)$ but $z \neq x$, then $S_3(z) = S_1(z) = S_2(z) = S_4(z)$. Hence $S_3(z) = S_4(z)$ for all $z \in M \neq x$. Also, $h(S_3) = (f(S_3) \text{ and } g(S_3)) = (\text{true or false}) = \text{true}$, while $h(S_4) = (f(S_4) \text{ and } g(S_4)) = (\text{false or false}) = \text{false} \neq h(S_3)$. Hence $x \in RR(h)$ by Definition 2.4, and $RR(f) \subseteq RR(h)$; by symmetry, $RR(g) \subseteq RR(h)$, so that $RR(f) \cup RR(g) \subseteq RR(h)$. □

The hypotheses of Theorem 2.6(b) are usually necessary; hence the conclusions might not follow if any of the following are false:

- $RR(f) \cap RR(g) = \emptyset$. Let $g = (\text{not } f)$ (that is, $g(S) = (\text{not } f(S))$ for all $S \in \mathbb{S}$). Then $h(S) = (f(S) \text{ or } (\text{not } f(S))) = \text{true}$ for all $S \in \mathbb{S}$, so that $h = \text{true}$ and $RR(h) = \emptyset$ by Theorem 2.2(c), even if $RR(f) \neq \emptyset$.
- $f \neq \text{true}$. If $f = \text{true}$, then $h(S) = (\text{true or } g(S)) = \text{true}$ for all $S \in \mathbb{S}$, so that $h = \text{true}$ and $RR(h) = \emptyset$, even if $RR(f) \neq \emptyset$.
- $g \neq \text{true}$. If $g = \text{true}$, then $h(S) = (f(S) \text{ or } \text{true}) = \text{true}$ for all $S \in \mathbb{S}$, so that $h = \text{true}$ and $RR(h) = \emptyset$, even if $RR(f) \neq \emptyset$.  

7
3 Conditional Relevant Regions

Suppose now that we have a partially defined C-expression \( f \), with a guard \( f^g \) as in Definition 2.13; that is, \( f(S) \) is defined whenever \( f^g(S) \) is true. We now ask whether there is a notion of a relevant region for \( f \), given its guard \( f^g \). Because \( f \) might actually be always defined, so that \( \text{RR}(f) \) is defined, we refer to this new notion as the conditional relevant region \( \text{CRR}(f, f^g) \). Note the explicit dependence on \( f^g \) here; the same C-expression might have different conditional relevant regions for different guards.

3.1 An Improper Relevant Region Definition

The definition of \( \text{CRR}(f, f^g) \) presents a difficulty much like the first of those presented for conditional input regions in Section 6 of [9]. We would like to require \( S_1 \) and \( S_2 \), as in Definition 2.4, to be such that \( f(S_1) \neq f(S_2) \). This would require that these both be defined, which would follow from \( f^g(S_1) = f^g(S_2) = \text{true} \). Unfortunately, that causes an unpleasant surprise, as we now show.

Example 3.1 Suppose we were to define our conditional relevant region as \( \{ x \in M : \text{there exist } S_1, S_2 \in \mathcal{S} \text{ such that } f^g(S_1) = f^g(S_2) = \text{true and } S_1(z) = S_2(z) \text{ for all } z \neq x, \text{ but } f(S_1) \neq f(S_2) \} \}. \) Now let \( f \) be \( v \) and let \( f^g \) be \( v = w \) (see Definitions 2.7 and 2.8 above). Here \( f \) clearly takes its value from \( v \) (or, equivalently, from \( w \)). However, we now show that this proposed conditional relevant region would not include \( v \). In fact, suppose the contrary; then there exist \( S_1 \) and \( S_2 \in \mathcal{S} \) such that \( f^g(S_1) = f^g(S_2) = \text{true} \) and \( S_1(z) = S_2(z) \) for all \( z \neq v \) (so that, in particular, \( S_1(w) = S_2(w) \)), but \( f(S_1) \neq f(S_2) \). Since \( f^g(S_1) = f^g(S_2) = \text{true} \), we have \( S_1(v) = S_1(w) = S_2(w) = S_2(v) \), by the definition of \( f^g \) in this case. But since \( S_1(z) = S_2(z) \) for all \( z \neq v \), we have, in fact, \( S_1 = S_2 \); so we cannot have \( f(S_1) \neq f(S_2) \).

There is no analogue, for relevant regions, of the conditionally executed instructions of section 5.4 of [9], and therefore no analogue of the second improper definition treated in section 6.1 of [9].

3.2 The Proper Definition

We now proceed to define \( \text{CRR}(f, f^g) \) in a way which is analogous to our Definition 2.12 for \( \text{CIR}(I, I^g) \). Just as \( \text{CIR}(I, I^g) \) contains \( \text{RR}(I^g) \), as a consequence of that definition, so \( \text{CRR}(f, f^g) \) contains \( \text{RR}(f^g) \) as a consequence of Definition 3.1 below; that is, the conditional relevant region of an expression
with a guard must include the relevant region of its guard. This fact will not be a part of the definition of CRR(f, f^\#); rather, it will be proved as part of Theorem 3.1 below. Informally, our definition says that any change in a variable in the conditional relevant region of f can either bring about a change in the value of f, or can change whether the guard of f is true or false.

**Definition 3.1** Under the conditions of Definition 2.13, the conditional relevant region CRR(f, f^\#) is defined as \{x \in M: there exist S_1 and S_2 \in S such that:

(a) S_1(z) = S_2(z) for all z \neq x;
(b) f^\#(S_1) = true; and
(c) either f^\#(S_2) = false, or else f(S_1) \neq f(S_2)\}.

Of the two parts of Definition 3.1(c), the first covers the case in which f(S_1) is known to be defined, but f(S_2) is not; while the second covers the case in which both f(S_1) and f(S_2) are known to be defined, but are unequal.

**Lemma 3.1** Using the notation of Definitions 2.4, 2.7, and 3.1:

(a) CRR(f, true) = RR(f).
(b) CRR(f, false) = \emptyset.

**PROOF.**

(a) This follows directly from the given definitions.
(b) If x \in CRR(f, false), then there exist S_1 and S_2 \in S such that all the conditions of Definition 3.1 hold. But condition (b), namely f^\#(S_1) = true, cannot hold (since f^\# = false); so x \notin CRR(f, false), and CRR(f, false) = \emptyset. □

We now prove a fundamental theorem for conditional relevant regions of C-expressions which corresponds to the fundamental theorem for ordinary relevant regions (Theorem 4 of [9]). As in [9], we will first need a definition, three lemmas, and a corollary.

**Definition 3.2** Let f : S \rightarrow T be a C-expression with guard f^\#, as in Definition 2.13, where S is the set of states of some computer with memory M. A subset M' of M is said to possess the conditional relevant region property CRP(f, f^\#) if RR(f^\#) \subseteq M' and if S_1|M' = S_2|M' implies f(S_1) = f(S_2), for all S_1, S_2 \in S for which f^\#(S_1) = f^\#(S_2) = true.

**Lemma 3.2** If M' and M'' are two subsets of M, each of which possesses CRP(f, f^\#), then M' \cap M'' also possesses CRP(f, f^\#).
PROOF. If \( RR(f^g) \subseteq M' \) and \( RR(f^g) \subseteq M'' \), then clearly \( RR(f^g) \subseteq M' \cap M'' \). Now let \( S_3, S_4 \in S \) be such that \( f^g(S_3) = f^g(S_4) = true \) and \( S_3|M \cap M'' = S_4|M' \cap M'' \); we need to show that \( f(S_3) = f(S_4) \). Let \( S_5 \) be defined by \( S_5(x) = S_3(x) \) for \( x \in M' \) and \( S_5(x) = S_4(x) \) for \( x \notin M' \). Here \( S_5 \in S \) by Axiom 1; and, since \( S_5|M' = S_3|M' \) and \( RR(f^g) \subseteq M' \), we have \( S_5|RR(f^g) = S_3|RR(f^g) \). We now have \( f^g(S_5) = f^g(S_3) \) (by Theorem 2.1) = true. We have \( S_5|M \setminus M' = S_4|M \setminus M' \); since \( M'' \subseteq M \), we have \( S_5|M'' \setminus M' = S_4|M'' \setminus M' \). Also, \( S_5|M' = S_3|M' \), and, since \( M' \cap M'' \subseteq M' \), we have \( S_5|M' \cap M'' = S_3|M' \cap M'' = S_4|M' \cap M'' \). Therefore \( S_5|(M'' \setminus M') \cup (M'' \cap M') = S_4|(M'' \setminus M') \cup (M'' \cap M') \). But \( (M'' \setminus M') \cup (M'' \cap M') = M'' \); so in fact \( S_5|M'' = S_4|M'' \). Since \( M' \) possesses CRP(\( f, f^g \)), and \( S_5|M' = S_3|M' \), and \( f^g(S_5) = f^g(S_3) = true \), we have \( f(S_5) = f(S_3) \) by Definition 3.2. Since \( M'' \) possesses CRP(\( f, f^g \)), and \( S_5|M'' = S_4|M'' \), and \( f^g(S_5) = f^g(S_3) = true \), we have \( f(S_5) = f(S_4) \), again by Definition 3.2. Therefore \( f(S_3) = f(S_4) \). □

Corollary 3.1 The intersection of any finite number of subsets of \( M \) possesses CRP(\( f, f^g \)) if each of these subsets possesses CRP(\( f, f^g \)).

This follows immediately by applying Lemma 3.2 successively. □

Lemma 3.3 If \( M' \) possesses CRP(\( f, f^g \)), and \( M'' \supseteq M' \), then \( M'' \) possesses CRP(\( f, f^g \)).

PROOF. If \( RR(f^g) \subseteq M' \) and \( M'' \supseteq M' \), then clearly \( RR(f^g) \subseteq M'' \). For any states \( S_1, S_2 \in S \) such that \( f^g(S_1) = f^g(S_2) = true \), let \( S_1|M'' = S_2|M'' \). Since \( M'' \supseteq M' \), we clearly have \( S_1|M' = S_2|M' \). By Definition 3.2, we have \( f(S_1) = f(S_2) \); but then, by the same definition, \( M'' \) possesses CRP(\( f, f^g \)). □

Lemma 3.4 CRR(\( f, f^g \)) = \{ \( x \in M : M \setminus \{ x \} \) does not possess CRP(\( f, f^g \)) \}.

PROOF. We will show that \( x \notin CRR(f, f^g) \) if and only if \( M \setminus \{ x \} \) possesses CRP(\( f, f^g \)). Let \( S_1, S_2 \in S \) be any two states such that \( S_1(x) = S_2(x) \) for all \( z \neq x \) (implying that \( S_1|M \setminus \{ x \} = S_2|M \setminus \{ x \} \)) and such that \( f^g(S_1) = true \). Now let \( M \setminus \{ x \} \) possess CRP(\( f, f^g \)). Here \( RR(f^g) \subseteq M \setminus \{ x \} \) by Definition 3.2, so that \( x \notin RR(f^g) \), which implies that \( f^g(S_1) = f^g(S_2) \) by Definition 2.4. Since \( f^g(S_1) = true \), we also have \( f^g(S_2) = true \), so that \( f(S_1) = f(S_2) \) by Definition 3.2; therefore \( x \notin CRR(f, f^g) \) by Definition 3.1.

Conversely, let \( x \notin CRR(f, f^g) \). By our assumptions, Definition 3.1(a) and (b) hold for \( S_1 \) and \( S_2 \); therefore, Definition 3.1(c) does not hold for these, implying that \( f^g(S_2) = true \) and \( f(S_1) = f(S_2) \). Since \( f^g(S_1) = true = f^g(S_2) \), we have \( x \notin RR(f^g) \) by Definition 2.4. Thus \( RR(f^g) \subseteq M \setminus \{ x \} \), and therefore \( M \setminus \{ x \} \) possesses CRP(\( f, f^g \)) by Definition 3.2. □
The following theorem now resembles Theorem 4 of [9] (and the two proofs are partially similar).

**Theorem 3.1** let \( f : \mathcal{S} \rightarrow T \) be a C-expression, with guard \( f^g : \mathcal{S} \rightarrow \{\text{true, false}\} \). Using the notation of Definition 2.13, for any states \( S_1, S_2 \in \mathcal{S} \), if \( f^g(S_1) = \text{true} \) and \( S_1|\text{CRR}(f,f^g) = S_2|\text{CRR}(f,f^g) \), then \( f^g(S_2) = \text{true} \) and \( f(S_1) = f(S_2) \). Also, \( \text{RR}(f^g) \subseteq \text{CRR}(f,f^g) \).

**PROOF.** Take the intersection \( M_1 \) of all subsets of \( M \) which possess \( \text{CRP}(f,f^g) \). Since each of these subsets contains \( \text{RR}(f^g) \), by Definition 3.2, their intersection \( M_1 \) does also. We first show that \( M_1 = \text{CRR}(f,f^g) \) (so that, in particular, \( \text{RR}(f^g) \subseteq \text{CRR}(f,f^g) \)). By Lemma 3.4, we need only show that, for each \( x \in M \), \( M \setminus \{x\} \) possesses \( \text{CRP}(f,f^g) \) if and only if \( x \notin M_1 \). But if \( M \setminus \{x\} \) possesses \( \text{CRP}(f,f^g) \), then \( M \setminus \{x\} \) is one of the subsets whose intersection is \( M_1 \), so that \( M_1 \subseteq M \setminus \{x\} \) and \( x \notin M_1 \). Conversely, if \( x \notin M_1 \), then there exists some set \( M_2 \) possessing \( \text{CRP}(f,f^g) \) to which \( x \) does not belong; and if \( M_2 \) possesses \( \text{CRP}(f,f^g) \), then since \( M \setminus \{x\} \supseteq M_2 \), \( M \setminus \{x\} \) possesses \( \text{CRP}(f,f^g) \) by Lemma 3.3.

Now let \( S_1, S_2 \in \mathcal{S} \) be such that \( S_1|\text{CRR}(f,f^g) = S_2|\text{CRR}(f,f^g) \) (that is, \( S_1|M_1 = S_2|M_1 \)), and such that \( f^g(S_1) = \text{true} \). Since \( S_1|\text{CRR}(f,f^g) = S_2|\text{CRR}(f,f^g) \), and since \( \text{RR}(f^g) \subseteq \text{CRR}(f,f^g) \) (as shown above), we have, in particular, \( S_1|\text{RR}(f^g) = S_2|\text{RR}(f^g) \). Therefore, \( f^g(S_2) = f^g(S_1) \) (by Definition 2.4) = \text{true}. By Axiom 2, \( \{x \in M : S_1(x) \neq S_2(x)\} \) is a finite set \( M_3 = \{x_1, \ldots, x_n\} \). Since \( S_1|M_1 = S_2|M_1 \), we have \( M_1 \cap M_3 = \emptyset \); therefore, each \( x_i \notin M_1 = \text{CRR}(f,f^g) \), so that \( M \setminus \{x_i\} \) possesses \( \text{CRP}(f,f^g) \) by Lemma 3.4. However, \( M \setminus M_3 \), being the intersection of all the \( n \) sets \( M \setminus \{x_i\} \), also possesses \( \text{CRP}(f,f^g) \) by Corollary 3.1. Since \( S_1|M \setminus M_3 = S_2|M \setminus M_3 \), and since \( f^g(S_1) = f^g(S_2) = \text{true} \), we thus have \( f^g(S_1) = f^g(S_2) \) by Definition 3.2. \( \square \)

There is an alternative, and easier, way of proving the last sentence of Theorem 3.1. If \( x \in \text{RR}(f^g) \), then there exist \( S_1 \) and \( S_2 \in \mathcal{S} \) such that \( S_1(z) = S_2(z) \) for all \( z \in M \) \( \neq x \), but \( f^g(S_1) \neq f^g(S_2) \). Since \( f^g(S_1) \) and \( f^g(S_2) \) are Boolean and unequal, one of them (which, by a change of notation if necessary, we can take to be \( f^g(S_1) \)) is \text{true}, while \( f^g(S_2) \) is \text{false}. But then \( x \in \text{CRR}(f,f^g) \) by Definition 3.1.

### 4 Partially Defined Expressions

In Chapter 7 of [9], we considered the conditional input and output regions of an instruction formed from two other instructions by composition. Here, in
an analogous way, we shall be concerned with the conditional relevant region of an expression with a guard, formed from two other expressions with guards by the application of an operator. Two complications arise here. The first is that expressions in a programming language might have side effects, which are beyond the scope of this paper. We now take up the other complication, namely that an operator might itself have a guard.

### 4.1 Guards and Relevant Regions of General Functions

The idea of a guard, as in Section 4.1 of [9], may be further extended to a general function of several variables. Suppose that \( f(x_1, \ldots, x_n) \) is in a set \( T \), where each \( x_i \) is assumed to be in some set \( B_i \); we may regard \( f \) as a function \( f : B_1 \times \cdots \times B_n \rightarrow T \). Suppose further that \( f(x_1, \ldots, x_n) \) is not always defined, but in fact will be defined if the \( x_i \) satisfy some condition \( f^g : B_1 \times \cdots \times B_n \rightarrow \{\text{true}, \text{false}\} \). Here, if \( f^g(x_1, \ldots, x_n) = \text{true} \), then \( f(x_1, \ldots, x_n) \) is defined. This is quite common in practice; for example, if \( f(x, y) = x/y \), then \( f^g(x, y) \) is \( y \neq 0 \).

Now consider a computer \((M, B, S, \mathbb{I})\), as in [8], in which \( M \) is the set \( \{1, \ldots, n\} \); \( B = B_1 \cup \cdots \cup B_n \); \( S \) is the set of all those functions \( S : M \rightarrow B \) such that \( S(i) \in B_i \) for each \( i, 1 \leq i \leq n \); and \( \mathbb{I} \) is empty. (Alternatively, we may regard \( S \) as the cartesian product of all the \( B_i \).) It should now be clear that \( f \) is a domain operation on \( S \), as in Definition 2.1 above; and that \( f^g \), as we have defined it, is the guard of \( f \), as in Definition 2.6 above. The conditional relevant region \( \text{CRR}(f, f^g) \) is then \( \{1, 2\} \), since the value of \( f \) depends on both of its parameters; the relevant region \( \text{RR}(f^g) \) of the guard of \( f \) is \( \{2\} \), since \( f^g \) depends only on its second parameter. For convenience later on, we specialize Definitions 2.2 and 3.1 as follows.

**Definition 4.1** Given a function \( f : B_1 \times \cdots \times B_n \rightarrow T \), its **relevant region** \( \text{RR}(f) \) is defined by \( \text{RR}(f) = \{k \in \{1, \ldots, n\} : \text{there exists } (x_1, \ldots, x_n) \in B_1 \times \cdots \times B_n \text{ such that } f(x_1, \ldots, x_n) \neq f(x_1, \ldots, x_{k-1}, x_k', x_{k+1}, \ldots, x_n)\}\).

**Definition 4.2** Given functions \( f : B_1 \times \cdots \times B_n \rightarrow T \) and \( f^g : B_1 \times \cdots \times B_n \rightarrow \{\text{true}, \text{false}\} \), such that \( f(x_1, \ldots, x_n) \) is defined whenever \( f^g(x_1, \ldots, x_n) = \text{true} \), the **conditional relevant region** \( \text{CRR}(f, f^g) \) is defined as \( \{k \in \{1, \ldots, n\} : \text{there exists } (x_1, \ldots, x_n) \in B_1 \times \cdots \times B_n \text{ and } x_k' \in B_k \text{ such that:}\)

(a) \( f^g(x_1, \ldots, x_n) = \text{true} \); and
(b) either \( f^g(x_1, \ldots, x_{k-1}, x_k', x_{k+1}, \ldots, x_n) = \text{false} \), or else \( f(x_1, \ldots, x_n) \neq f(x_1, \ldots, x_{k-1}, x_k', x_{k+1}, \ldots, x_n)\)\).

Each \( S \in S \) may be thought of as an \( n \)-tuple \((x_1, \ldots, x_n)\), where \( S(i) = x_i \) for
each \(i, 1 \leq i \leq n\). A function \(f(x_1, \ldots, x_n)\) of \(n\) variables can be thought of as a function \(f\) of the \(n\)-tuple \((x_1, \ldots, x_n)\), and hence as a function \(f(S)\), where \(S\) is this \(n\)-tuple.

### 4.2 Guards of Operators

Of particular interest, in this connection, are the guards of operators. In particular, we may consider \(f(x, y) = x/y\) as the mathematical definition of the ordinary division operator. We now recall the distinction made in [9] between ideal and actual type-sets. The type-set \(B_x\) of a variable \(x \in M\) is the set of all allowable values of \(x\), and an actual type-set for \(x\) is a type-set with restrictions imposed by an actual computer, while its ideal type-set has no such restrictions. For example, if \(x\) is a \(d\)-bit (signed) twos’ complement integer variable, its ideal type-set is the set of all integers, whereas its actual type-set is the set of all integers in the range from \(-2^{d-1}\) through \(2^{d-1} - 1\). (Recall that in this case we can have \(x = -2^{d-1}\) but we cannot have \(x = 2^{d-1}\), due to the way that twos’ complement arithmetic works.) Similarly, if \(x\) is a real variable, its actual type-set is the set of all floating-point numbers on some specific computer, whereas its ideal type-set is the set of all real numbers.

There is a great difference between the ideal and the actual cases with respect to the guards of operators. In the actual case, there is always a basic \textit{inrange} function which guarantees that its argument is in range. Thus:

- For \(d\)-bit unsigned arithmetic, \textit{inrange}(\(x\)) is \((x \geq 0 \text{ and } x < 2^d)\).
- For \(d\)-bit twos’ complement signed arithmetic, \textit{inrange}(\(x\)) is \((-2^{d-1} \leq x \text{ and } x < 2^{d-1})\).

The guard of \(B(x, y)\), for the binary operator \(B\), is then \((\text{inrange}(x) \text{ and inrange}(y) \text{ and inrange}(B(x, y)))\); the guard of \(U(x)\), for the unary operator \(U\), is \((\text{inrange}(x) \text{ and inrange}(U(x)))\). For example, for \(x + y\), we must have \(x, y\), and \(x + y\) all in range; for \(-x\), we must have \(x\) and \(-x\) both in range. (This is not automatic, for twos’ complement arithmetic; thus if \(x = -2^{d-1}\), then \(x\) is in range, but \(-x = 2^{d-1}\) is not.) The floating point \textit{inrange}(\(x\)) function, which specifies that \(x\) is a legal floating point number, is much more complex; also, most floating point computations are approximate, and this is all that can be proved about them.

Now consider the expression \(f(e_1, \ldots, e_n)\), where each expression \(e_i\) has its own guard \(e_i^g\). In order for \(f(e_1, \ldots, e_n)\) to be defined, each \(e_i\) must be defined; that is, each \(e_i^g\) must be \textit{true}. In addition, however, \(f\) must also be defined on the values of \(e_1, \ldots, e_n\).

**Example 4.1** Consider the expression \((A/B)/(C/D)\). Writing this as \(f(f(A,
B), \( f(C, D) \), where \( f(x, y) = x/y \), we may calculate its guard as \( f^g(A, B) \) (that is, \( B \neq 0 \)) and \( f^g(C, D) \) (that is, \( D \neq 0 \)) and \( f^g(f(A, B), f(C, D)) \) (that is, \( f(C, D) \neq 0 \), or \( C/D \neq 0 \)).

We now generalize this as follows.

**Definition 4.3** Given \( n \) domain operations \( f_i : S \to T_i \), each with a guard \( f^g_i : S \to \{ \text{true, false} \} \) (that is, \( f_i(S) \) is defined if \( f^g_i(S) = \text{true} \)), and a function \( f : T_1 \times \cdots \times T_n \to T \), for some set \( T \), with a guard \( f^g : T_1 \times \cdots \times T_n \to \{ \text{true, false} \} \) (that is, \( f(x_1, \ldots, x_n) \) is defined if \( x_i \in T_i \) for \( 1 \leq i \leq n \) and \( f^g(x_1, \ldots, x_n) = \text{true} \)), the **f-term** \( \tau(f, f_1, \ldots, f_n) = t : S \to T \) and its guard \( t^g : S \to \{ \text{true, false} \} \), are defined by \( t^g(S) = \text{true} \), for \( S \in S \), if and only if:

- \( f^g(S) = \text{true} \) (so that \( f_i(S) \) is defined) for all \( i, 1 \leq i \leq n \), and
- \( f^g(f_1(S), \ldots, f_n(S)) = \text{true} \) (so that \( f(f_1(S), \ldots, f_n(S)) \) is defined)

and \( t(S) = f(f_1(S), \ldots, f_n(S)) \) whenever \( t^g(S) = \text{true} \).

The functions \( f \) corresponding to the five basic arithmetic operations (+, −, \*, /, and unary −) may be defined as

- \( \text{add}(x, y) = x + y \)
- \( \text{sub}(x, y) = x - y \)
- \( \text{mul}(x, y) = xy \)
- \( \text{div}(x, y) = x/y \)
- \( \text{neg}(x) = -x \)

and their corresponding terms are add-terms, sub-terms, mul-terms, div-terms, and neg-terms, respectively. (All this is done in the ideal case only, in which the value of any variable is allowed to be an arbitrary real number. In the actual case, the notation would be ambiguous; the sum of two variables, for example, would have different meanings depending on whether they are signed or unsigned, two-byte or four-byte, and the like.) Likewise, we introduce:

- \( \text{eq}(x, y) = \text{true} \) if \( x = y \) and false otherwise, with corresponding eq-terms, and similarly for the other relational operators (ne, gt, lt, ge, le) and their corresponding terms; and
- \( \text{and}(x, y) = x \) and \( y \), with corresponding and-terms, and similarly for the other logical operators (or, xor, not) and their corresponding terms.

We may use Definition 4.3 repeatedly to form domain operations from larger expressions; and we adopt a notation for these, which extends that of Section 5 of [9]. Given an expression \( e \) in the algebraic-language sense, such as \( b * b - 4 * a * c \), we define its corresponding domain operation \( f \), in this case \( b * b - 4 * a * c \), as \( f(S) = e' \) where \( e' \) is obtained from \( e \) by replacing each variable name \( \alpha \) by \( S(\alpha) \). In this case we would have \( f(S) = S(b) * S(b) - 4 * S(a) * S(c) \) for
each \( S \in \mathbb{S} \) (remember that only variable names, not constants, are replaced in this way). We will use \( +, -, \ast, /, =, \neq, >, \leq, \geq \), and \( \textbf{and}, \textbf{or}, \textbf{xor} \), and \textbf{not} in such domain operations, with their usual meanings. The precedences of all these operators are taken to be as in the C language.

4.3 Relevant Regions of \( f \)-Term Guards

Using the notation of Definition 4.3, we can now derive a relation between the relevant region of the guard of \( t \) and the relevant regions of the various \( f_i \) and their guards. In order to understand the general case, it helps to consider \( t = \frac{A}{B} / C / D \), much as in Example 4.1. Here \( f_1 \) is \( A / B \); \( f_1^0 \) is \( (B \neq 0) \); \( f_2 \) is \( C / D \); \( f_2^0 \) is \( (D \neq 0) \); \( f(x, y) = x / y \); and \( f^0(x, y) = y \neq 0 \). Clearly we should have \( t^0 = (B \neq 0) \) and \( (D \neq 0) \), and therefore \( RR(t^0) = \{B, C, D\} \). Here \( RR(t^0) \) includes \( RR(f_1^0) = \{B\} \) and \( RR(f_2^0) = \{D\} \); it also includes \( CRR(f_2, f_2^0) = \{C, D\} \), but not \( CRR(f_1, f_1^0) = \{A, B\} \). Informally, this is because \( f^0(x, y) \) depends on its second argument, but not on its first.

Formally, we consider \( RR(f^0) \), which, by Definition 4.1, is a subset of \( \{1, \ldots, n\} \). If \( j \notin RR(f^0) \), then \( f^0 \) does not depend on its \( j \)th argument, and we need not consider \( CRR(f_j, f_j^0) \) (just as in the preceding example, where we had \( j = 1 \)). In other words, we need to consider \( CRR(f_j, f_j^0) \) only for those values of \( j \) which are in \( RR(f^0) \). This leads to the following.

**Theorem 4.1** Using the notation of Definition 4.3, we have:

\[
RR(t^0) \subseteq \bigcup_{i=1}^{n} RR(f^0_i) \cup \bigcup_{j \in RR(f^0)} CRR(f_j, f^0_j)
\]

**PROOF.** Let \( x \notin \bigcup_{i=1}^{n} RR(f^0_i) \) and \( x \notin \bigcup_{j \in RR(f^0)} CRR(f_j, f^0_j) \), so that \( x \notin RR(f^0_i) \) for all \( i, 1 \leq i \leq n \), and also \( x \notin CRR(f_j, f^0_j) \) for all \( j \in RR(f^0) \). By Theorem 2.1, given any \( S_1 \) and \( S_2 \in \mathbb{S} \) such that \( S_1(z) = S_2(z) \) for all \( z \in M \) \( \neq x \), we thus have \( f^0_i(S_1) = f^0_i(S_2) \) for all \( i, 1 \leq i \leq n \). We will show that \( t^0(S_1) = t^0(S_2) \), so that \( x \notin RR(t^0) \) by Definition 2.4; and this will immediately imply our conclusion. To do this, it suffices to show that, if \( t^0(S_1) = true \), then \( t^0(S_2) = true \). This is because then, if \( t^0(S_2) = true \), then \( t^0(S_1) = true \) by symmetry, so that, if \( t^0(S_1) = false \), then \( t^0(S_2) = false \) (note that \( t^0 \), being a guard, is everywhere defined). Thus, in either case, we will have \( t^0(S_1) = t^0(S_2) \). Suppose, then, that \( t^0(S_1) = true \). For all \( i, 1 \leq i \leq n \), we therefore have, by Definition 4.3, that \( f^0_i(S_1) = true \) (so that \( f_i(S_1) \) is defined), and \( f^0_i(S_1), \ldots, f_n(S_1) = true \). However, as we have seen, \( f^0_i(S_1) = f^0_i(S_2) \), so that \( f^0_i(S_2) = true \) (and \( f_i(S_2) \) is defined).
It remains only to show that \( t^g(S_2) = f^g(f_1(S_2), \ldots, f_n(S_2)) = f^g(f_1(S_1), \ldots, f_n(S_1)) \) = \text{true}. For \( 0 \leq i \leq n \), we define \( z_i = f^g(f_1(S_1), \ldots, f_i(S_1), f_{i+1}(S_2), \ldots, f_n(S_2)) \); in particular, \( z_0 = f^g(f_1(S_2), \ldots, f_n(S_2)) \) and \( z_n = f^g(f_1(S_1), \ldots, f_n(S_1)) \). It suffices, then, to show that \( z_0 = z_n \); to do this, we show that \( z_{j-1} = z_j \) for all \( j, 1 \leq j \leq n \) (so that \( z_0 = z_1 = \cdots = z_n \)). By definition of the \( z_i \), the arguments of \( f^g \) in \( z_{j-1} \) and in \( z_j \) differ only at position \( j \) (at most). If \( j \notin RR(f^g) \), therefore, we have \( z_{j-1} = z_j \) directly from Definition 4.1. If \( j \in RR(f^g) \), then \( x \notin CRR(f_j, f^g_j) \) by hypothesis; and, since \( f^g_j(S_1) = \text{true} \), we have \( f_j(S_1) = f_j(S_2) \) by Definition 3.1. Thus the arguments of \( f^g \) in \( z_{j-1} \) and in \( z_j \) do not differ even at position \( j \), which shows that \( z_{j-1} = z_j \). \( \square \)

**Corollary 4.1** If \( f^g = \text{true} \) (that is, \( f \) has no guard), then:

\[
RR(t^g) \subseteq \bigcup_{i=1}^{n} RR(f_i^g)
\]

**PROOF.** Since \( f^g = \text{true} \), we have \( RR(f^g) = \emptyset \) by Theorem 2.2(c). Since there are no \( j \in RR(f^g) \), we have \( \bigcup_{j \in RR(f^g)} CRR(f_j, f^g_j) = \emptyset \). Thus the corollary follows immediately from the theorem. \( \square \)

### 4.4 Relevant Regions of \( f \)-Terms Themselves

We now derive a relation between the conditional relevant region of \( t \) and those of the various \( f_i \). The form of this theorem, presented as Theorem 4.2 below, might seem unexpected.

**Example 4.2** Suppose that \( n = 2 \); \( e_1 = u \); \( e_1^g = \text{true} \); \( e_2 = 1/v \); \( e_2^g = v \neq 0 \); \( f(x, y) = x \); and \( f^g(x, y) = \text{true} \). Note that \( CRR(f, f^g) = \{1\} \) here, since the result does not depend on argument 2; also, \( f(e_1, e_2) = u \), which does not involve \( v \) at all. Are we to conclude, then, much as in Theorem 4.1, that \( CRR(t, t^g) \) is contained in the union of only those \( CRR(f_j, f^g_j) \) such that \( j \in CRR(f, f^g) \) — so that, in particular, \( v \notin CRR(t, t^g) \)? This might look plausible, but it doesn’t hold. Let \( S_1 \) be such that \( t^g(S_1) = \text{true} \); then, by Definition 4.3, we must, in particular, have \( e_2^g(S_1) = \text{true} \), so that \( S_1(v) \neq 0 \). If \( S_2 \) is defined by \( S_2(z) = S_1(z) \) for all \( z(\in M) \neq v \), but \( S_2(v) = 0 \), then \( e_2^g(S_2) = \text{false} \), so that, necessarily, \( t^g(S_2) = \text{false} \). Hence the conditions of Definition 3.1 are satisfied, and \( v \in CRR(t, t^g) \).

The form of Theorem 4.2 below is accordingly simpler than that of Theorem 4.1.
Theorem 4.2 Using the notation of Definition 4.3, we have \( CRR(t, t^g) \subseteq \bigcup_{i=1}^n CRR(t_i, f_i^g) \).

PROOF. Let \( x \notin \bigcup_{i=1}^n CRR(f_i, f_i^g) \), so that \( x \notin CRR(f_i, f_i^g) \) for all \( i, 1 \leq i \leq n \). Let \( S_1, S_2 \in \mathbb{S} \) be such that \( S_1(z) = S_2(z) \) for all \( z (\in M) \neq x \), and such that \( t^g(S_1) = true \). Thus, by Definition 4.3, we have that \( f_i^g(S_1) = true \) (so that \( f_i(S_1) \) is defined) for all \( i, 1 \leq i \leq n \), as well as that \( f^g(f_1(S_1), \ldots, f_n(S_1)) = true \) (so that \( f(f_1(S_1), \ldots, f_n(S_1)) \) is defined). For \( 1 \leq i \leq n \), by Definition 3.1, since \( x \notin CRR(f_i, f_i^g) \), we therefore have \( f_i^g(S_2) = true \) (so that \( f_i(S_2) \) is defined) and \( f_i(S_1) = f_i(S_2) \). Therefore, \( f^g(f_1(S_2), \ldots, f_n(S_2)) = f^g(f_1(S_1), \ldots, f_n(S_1)) = true \), so that \( f(f_1(S_2), \ldots, f_n(S_2)) \) is defined. Hence \( t(S_1) = f(f_1(S_1), \ldots, f_n(S_1)) = f(f_1(S_2), \ldots, f_n(S_2)) = t(S_2) \), so that \( x \notin CRR(t, t^g) \) by Definition 3.1, and \( CRR(t, t^g) \subseteq \bigcup_{i=1}^n CRR(f_i, f_i^g) \). \( \square \)

Corollary 4.2 Using the notation of Definition 4.3, if \( t \) has no guard, then \( RR(t) \subseteq \bigcup_{i=1}^n RR(f_i) \).

PROOF. To say that \( t \) has no guard means that \( t^g = true \), so that \( CRR(t, t^g) = CRR(t, true) = RR(t) \) by Lemma 3.1(a). Then, for all \( S \in \mathbb{S} \) and all \( i, 1 \leq i \leq n \), since \( t^g(S) = true \), we have \( f_i^g(S) = true \) by Definition 4.3, and thus \( CRR(f_i, f_i^g) = CRR(f_i, true) = RR(f_i) \), again by Lemma 3.1(a). The corollary now follows immediately from the theorem. \( \square \)

Partial converses to Theorems 4.1 and 4.2 may be constructed; but these are unexpectedly complex, and are beyond the scope of this paper.

4.5 Subscripts and Pointers on Left Sides of Assignments

So far we have considered only assignments to a simple variable. Some assignments, however, have more general left sides, such as subscripted variables or pointers. We may associate, with any left side, a C-expression \( L \) such that, if \( S \) is the current state vector, then \( L(S) \) is the variable currently specified by \( L \). This variable is in \( M \), the memory of the computer, and we refer to \( L \) as a selector, having a memory footprint which is the set of all the variables which it can specify. Every left side \( L \) has a corresponding right side \( R \), such that \( R(S) \) is the current value of \( L(S) \). As with other expressions, left sides with side effects are not considered here.

Definition 4.4 Given a computer \( \langle M, B, \mathbb{S}, I \rangle \) or \( \langle M, \mathbb{S}, I \rangle \), the selector associated with the left side of an assignment is a C-expression \( L : \mathbb{S} \rightarrow M \), possibly with a guard \( L^g \); here \( L(S) \) (\( \in M \)) is the variable currently specified
by this left side, if \( S \) is the current state. The memory footprint \( \mu(L, L^g) \)
is the set of all \( u \in M \) such that there exists \( S \in S \) with \( L^g(S) = \text{true} \)
and \( L(S) = u \); we write \( \mu(L) \) as an abbreviation for \( \mu(L, \text{true}) \). The right side
\( R \) associated with \( L \) is such that \( R^g = L^g \) and \( R(S) = S(L(S)) \).

To find the value of \( R(S) \), then, we first find the value of \( L(S) \) (that is, the
variable \( v \) currently associated with \( L \)) and then find the current value \( S(v) \),
that is, \( S(L(S)) \). If a left side is a simple variable \( v \), its corresponding selector
\( L \) is constant. That is, \( L(S) \equiv v \) for all \( S \in S \); \( \mu(L) = \{v\} \); and \( R(S) = S(v) \).

Example 4.3 Let \( e \) be a C-expression with integer values, and let \( t \) be an
array defined as having elements from \( t[0] \) through \( t[n-1] \). (The parameter
\( n \), here, is assumed to be independent of the state of the computer — it is
either not in \( M \) at all, or else has only one value.) Here \( t \) is a subset of \( M \);
that is, each of the \( n \) elements of \( t \) is, separately, an element of \( M \). We may
now associate with \( t[e] \) a selector \( L : S \to M \) such that \( L(S) = t[e(S)] \) for any
state \( S \in S \). The guard \( L^g(S) \) is true if and only if \( 0 \leq e(S) \leq n-1 \) (meaning
that \( t[e] \) is in range); and the memory footprint \( \mu(L, L^g) \) is \( t \). The right side \( R \)
associated with \( L \) now corresponds to \( t[e] \) as a right-side expression, meaning
the current value of \( t[e] \).

Example 4.4 Let \( p \) be a pointer variable, so that its values are themselves
variables; that is, elements of \( M \). The left side \( \ast p \) (as in C or C++) may now
be associated with the selector \( L : S \to M \) such that \( L(S) = S(p) \), which is in
\( M \). The right side \( R \) associated with \( L \) now corresponds to \( \ast p \) as a right-side
expression, meaning the current value of the element to which \( p \) points. If that
value is known to be in some set \( T \) (as required by the declaration of \( p \) in C
or C++), then we may write \( R : S \to T \) for the right side \( R \). If \( L \) is taken to
have no guard, the memory footprint \( \mu(L) \) might be the entire memory \( M \).

The calculated memory footprint of a selector \( L \) should normally include only
those variables which might be changed, in the actual situation. We can often
bring this about by a choice of guard for \( L \).

Example 4.5 In Example 4.3 above, the actual memory footprint of \( L \) might
not be \( t[0] \) through \( t[n-1] \), but rather \( t[0] \) through \( t[k-1] \), where \( k < n \). This
will be \( \mu(L, L^g) \), where the guard \( L^g \) of \( L \) is chosen to be such that \( L^g(S) = \text{true} \)
if and only if \( 0 \leq e(S) \leq k-1 \).

Example 4.6 In Example 4.4 above, the variables that can be changed by
setting \( \ast p \) do not, in practice, range over the entire memory, but only over
some array \( t \), where \( p \) is assumed to point to some element of \( t \). We then
choose \( L^g \) to be such that \( L^g(S) = \text{true} \) if and only if \( S(p) \) is an element of \( t \);
and, as before, the memory footprint \( \mu(L, L^g) \) is \( t \).
In [9] (Definition 13), we defined the instruction $v \leftarrow f$, where $v \in M$ and $f : S \rightarrow B_v$ is a C-expression. We now generalize this to the case in which $v$ is replaced by a more general selector $e$, where both $e$ and $f$ might have guards (still assuming, as in [9], no conversion from one type-set to another).

**Definition 4.5** If $e : S \rightarrow M$ is a selector, as in Definition 4.4, with guard $e^g$, and if $f : S \rightarrow B_v$ is a C-expression, with guard $f^g$, then $e \leftarrow f$ is the instruction $I$ such that $I(S) = S'$ where $S'(e(S)) = f(S)$ and $S'(z) = S(z)$ for all $z \in M \neq e(S)$. The guard of $I$ is defined by $I^g(S) = (e^g(S) \text{ and } f^g(S))$.

If $e$ is a constant selector, $e(S) \equiv v$ for all $S \in S$, and if neither $e$ nor $f$ has a guard, then $e \leftarrow f$ reduces to $v \leftarrow f$, as in [9].

The conditional output region of $e \leftarrow f$ is contained in the memory footprint of $e$. The conditional input region of $e \leftarrow f$ might seem surprising. It clearly involves the conditional relevant regions of both $e$ and $f$ (for example, an instruction setting $t[k]$ equal to $x$ will have both $k$ and $x$ in its input region). It can also, however, involve $\mu(e,e^g)$. In particular, setting a variable element of an array, such as $t[k] = x$, where the array $t$ contains at least two elements, normally involves the entire array $t$ in the input region. Here $t$ is clearly in the output region, since any element of $t$ might be changed.

Now (for example), $t[1]$ is in the input region. In fact, let $S$ and $S' \in S$ be such that $S'(t[1]) \neq S(t[1])$ and $S'(z) = S(z)$ for all $z \in M \neq t[1]$. Furthermore let $S(k) = 2$ (here we use the fact that $t$ contains at least two elements), so that $I(S)(t[1]) = S(t[1])$ and $I(S')(t[1]) = S'(t[1])$. Then there does exist an element $y$ of the output region (namely $y = t[1]$ itself), such that $I(S')(y) = I(S')(t[1]) = S'(t[1]) \neq S(t[1]) = I(S)(t[1]) = I(S)(y)$! In formally, we can see this, for an array of size 2, by rewriting $t[k] = x$ as $t[1] = (\text{if } k=1 \text{ then } x \text{ else } t[1])$ followed by $t[2] = (\text{if } k=2 \text{ then } x \text{ else } t[2])$, which makes it clearer that $t[1]$ and $t[2]$ are in the input region. Similar arguments work for larger arrays.

We can show that $\text{CIR}(I, I^g) \subseteq \text{CRR}(e, e^g) \cup \text{CRR}(f, f^g) \cup \mu(e, e^g)$; it is possible, however, to derive a stronger condition (see Theorem 4.3(b) below). The fact that this is stronger might seem obvious; however, despite what might be suggested by Lemma 3.1, a more restrictive guard does not always lead to a smaller conditional relevant region. For example, $\text{CRR}(y = x, \text{true})$ is $\{x, y\}$, but $\text{CRR}(y = x, z \neq 0)$ is $\{x, y, z\}$, which is larger, not smaller. Therefore, we use a slightly longer argument, as follows.

**Lemma 4.1** Let $f$ be as in Definition 4.5; let $g_1$ and $g_2$ be guards of $f$; and let $g_3$ be defined by $g_3(S) = g_1(S) \text{ and } g_2(S)$, for all $S \in S$. Then $\text{CRR}(f, g_3) \subseteq \text{CRR}(f, g_1) \cup \text{RR}(g_2)$. 

19
PROOF. Let $x \in \text{CRR}(f, g_3)$, so that, by Definition 3.1, there exist $S_1$ and $S_2 \in \mathbb{S}$ such that $S_1(z) = S_2(z)$ for all $z \neq x$; $g_3(S_1) = \text{true}$ (and therefore $g_1(S_1) = g_2(S_1) = \text{true}$); and either $g_3(S_2) = \text{false}$, or else $f(S_1) \neq f(S_2)$ (in which case $x \in \text{CRR}(f, g_1)$ follows immediately from Definition 3.1). If $g_3(S_2) = \text{false}$, then either $g_1(S_2)$ or $g_2(S_2)$ must be $\text{false}$, by the definition of $g_3$. But if $g_1(S_2) = \text{false}$, then $x \in \text{CRR}(f, g_1)$ by Definition 3.1; while if $g_2(S_2) = \text{false}$, then $x \in \text{RR}(g_2)$ by Definition 2.4, since $g_2(S_1) = \text{true}$. □

**Lemma 4.2** Using the terminology of Definition 4.4 and Lemma 4.1, we have $\mu(L, g_3) \subseteq \mu(L, g_1)$.

PROOF. Let $x \in \mu(L, g_3)$, so that, by Definition 4.4, there exists $S \in \mathbb{S}$ with $g_3(S) = \text{true}$ and $L(S) = u$. By the hypothesis of Lemma 4.1, since $g_3(S) = \text{true}$, we have $g_1(S) = \text{true}$. But then $x \in \mu(L, g_1)$, again by Definition 4.4. □

**Lemma 4.3** Using the terminology of Definition 4.4, we have $\text{CRR}(e, e^g \text{ and } f^g) \cup \text{CRR}(f, e^g \text{ and } f^g) \cup \mu(e, e^g \text{ and } f^g) \subseteq \text{CRR}(e, e^g) \cup \text{CRR}(f, f^g) \cup \mu(e, e^g)$.

PROOF. By Lemma 4.1, we have $\text{CRR}(e, e^g \text{ and } f^g) \subseteq \text{CRR}(e, e^g) \cup \text{RR}(f^g)$ and $\text{CRR}(f, f^g \text{ and } e^g) \subseteq \text{CRR}(f, f^g) \cup \text{RR}(e^g)$; so $\text{CRR}(e, e^g \text{ and } f^g) \cup \text{CRR}(f, e^g \text{ and } f^g) \subseteq \text{CRR}(e, e^g) \cup \text{RR}(f^g) \cup \text{CRR}(f, f^g) \cup \text{RR}(e^g)$. However, the right side of this reduces to $\text{CRR}(e, e^g) \cup \text{CRR}(f, f^g)$, since $\text{RR}(e^g) \subseteq \text{CRR}(e, e^g)$ and $\text{RR}(f^g) \subseteq \text{CRR}(f, f^g)$ by the last part of Theorem 3.1. Also, $\mu(e, e^g \text{ and } f^g) \subseteq \mu(e, e^g)$ by Lemma 4.2. The present lemma follows immediately. □

**Theorem 4.3** If $I$ is $e \leftarrow f$, as in Definition 4.5, then:

(a) $\text{COR}(I, I^g) \subseteq \mu(e, e^g \text{ and } f^g)$.
(b) $\text{CIR}(I, I^g) \subseteq \text{CRR}(e, e^g \text{ and } f^g) \cup \text{CRR}(f, e^g \text{ and } f^g) \cup \mu(e, e^g \text{ and } f^g)$.

PROOF.

(a) Let $S$ be such that $I^g(S) = \text{true}$, so that, by Definition 4.5, we have $e^g(S) = \text{true}$ and $f^g(S) = \text{true}$. Let $S' = I(S)$, and suppose that $z \notin \mu(e, e^g \text{ and } f^g)$. By the definition of this (in Definition 4.4), we have $e(S) \neq z$; and thus $S'(z) = S(z)$ by Definition 4.5. Since this holds for all $S$ with $I^g(S) = \text{true}$, we have $z \notin \text{COR}(I, I^g)$ by Definition 2.10, so that $\text{COR}(I, I^g) \subseteq \mu(e, e^g \text{ and } f^g)$.
(b) Let \( x \notin \text{CRR}(e, e^g \text{ and } f^g) \) and let \( x \notin \text{CRR}(f, e^g \text{ and } f^g) \). By Definition 3.1, this means that, if \( S_1 \) and \( S_2 \in \mathbb{S} \) are such that \( S_1(z) = S_2(z) \) for all \( z \in M \), and if \( e^g(S_1) = \text{true} \) and \( f^g(S_1) = \text{true} \) (implying that \( I^g(S_1) = \text{true} \), by Definition 4.5), we must have \( e^g(S_2) = \text{true} \) and \( f^g(S_2) = \text{true} \) (implying that \( I^g(S_2) = \text{true} \), by Definition 4.5); \( e(S_1) = e(S_2) \); and \( f(S_1) = f(S_2) \). If we had \( x \in \text{CIR}(I, I^g) \), then, for some such \( S_1 \) and \( S_2 \), there would have to be \( y \in \text{COR}(I, I^g) \) with \( I(S_1)(y) \neq I(S_2)(y) \), by Definition 2.12(c) (since \( I^g(S_2) \neq \text{false} \)). Since \( \text{COR}(I, I^g) \subseteq \mu(e, e^g \text{ and } f^g) \), we would have \( y \in \mu(e, e^g \text{ and } f^g) \). Now let \( x \notin \mu(e, e^g \text{ and } f^g) \), so that clearly \( y \neq x \); there are then two cases. If \( e(S_1) (= e(S_2)) = y \), then \( I(S_1)(y) = I(S_1)(e(S_1)) = f(S_1) = f(S_2) = I(S_2)(e(S_2)) = I(S_2)(y) \), a contradiction. If \( e(S_1) (= e(S_2)) \neq y \), then \( I(S_1)(y) = S_1(y) = S_2(y) \) (since \( y \neq x \) = \( I(S_2)(y) \), again a contradiction. Hence \( x \notin \text{CIR}(I, I^g) \), and \( \text{CIR}(I, I^g) \subseteq \text{CRR}(e, e^g \text{ and } f^g) \cup \text{CRR}(f, e^g \text{ and } f^g) \cup \mu(e, e^g \text{ and } f^g) \).

5 Partially Defined Sequential Programs

We now consider sequential programs, made up of statements in sequence. Each of these statements has an associated instruction, which we call its effect, and also an associated integer expression, which we call its controller. The value of the controller determines what statement is to be executed next after this one, if any. Our definition here extends that of Example 3 of [9] in two ways. First, effects and controllers can have guards; second, our statements have indices in a general range from \( t \) through \( t + n - 1 \), rather than always from 1 through \( n \). This is so that a contiguous section of a sequential program can also be a sequential program (see Section 5.1 below). As in Example 3 of [9], however, we consider only single-entry, single-exit programs, since these are the only ones to which the methods of this paper apply. Two parts of a program \( P \) cannot be interchanged, for example, if one of them has an alternate exit which might go to a different statement of \( P \) (also see Section 5.1 below).

**Definition 5.1** Let \( \mathbb{S} \) be the set of states of a computer. On this computer:

(a) A sequential program, with start index \( t \) and length \( n > 0 \), is a sequence \( P \) of \( n \) statements, \( P_t, P_{t+1}, \ldots, P_{t+n-1} \), such that, with each \( P_i \), there is associated an effect \( e_i : \mathbb{S} \to \mathbb{S} \), with a guard \( e^g_i : \mathbb{S} \to \{\text{true}, \text{false}\} \), and a controller \( c_i : \mathbb{S} \to \{t, \ldots, t+n\} \), with a guard \( c^g_i : \mathbb{S} \to \{\text{true}, \text{false}\} \). Here if \( S \) is the current state, then \( e_i(S) \) is the new current state. If \( c_i(S) = j \leq t+n-1 \), then \( P_j \) is the next statement to be executed after \( P_i \); whereas, if \( c_i(S) = t+n \), then \( P_i \) is the last statement of \( P \). (As usual, although we specify \( \mathbb{S} \) as the domain of both \( e_i \) and \( c_i \), each of these will be taken as undefined for those \( S \) for which its guard is false.)
(b) A sequential statement is a statement \( P_i \) of a sequential program \( P \), such that \( (c_i(S), e_i^g(S)) = (i + 1, \text{true}) \) (see Definition 2.14). (Note that if \( i \neq t + n - 1 \), control passes to the next statement of \( P \); while, if \( i = t + n - 1 \), this is the last statement of \( P \).)

(c) For any statement \( P_i \), if \( S \) is the current state, and \( (e_i(S), e_i^g(S)) = (S, \text{true}) \) for all \( S \in \mathcal{S} \), then we say that \( P_i \) has the null effect. If \( c_i(S) = k \), we say that \( P_i \) branches to statement \( k \), unless \( k = i + 1 \), in which case we say that \( P_i \) does not branch, or unless \( k = t + n \), in which case we say that \( P_i \) exits.

(d) A simple m-way branch is a statement \( P_i \) of a sequential program \( P \), having the null effect (see part (c) above), and such that \( c_i \) has exactly \( m \) distinct values, for some \( m \geq 1 \). A simple unconditional branch is a simple 1-way branch. (We call such a branch simple, meaning that it does not have side effects, since it has the null effect.)

(e) The directed graph of \( P \), as in (a) above, has vertices \( t \) through \( t + n \), with an edge from \( i \) to \( j \) (where \( t \leq i \leq t + n - 1 \) and \( t \leq j \leq t + n \)) if and only if there is a state \( S \in \mathcal{S} \) with \( c_i(S) = j \).

Besides each statement of a sequential program having an effect with a guard, the entire program \( P \) has an effect \( e_P \) with a guard \( e_P^g \), as defined below. If \( S \) is the state when \( P \) is started, then \( e_P(S) \) is the state when the program exits. Here \( e_P^g(S) \) is true only when the program actually exits (that is, it does not loop endlessly, or try to execute a statement which is undefined at the time).

Definition 5.2 Let \( P \) be a sequential program as in Definition 5.1(a) above; then we define its effect \( e_P \), with guard \( e_P^g \), as follows. Let \( S \in \mathcal{S} \), and construct an execution sequence \( (S_0, Q_0) = (S,t), (S_1, Q_1), (S_2, Q_2), \ldots \), as follows, where \( S_i \in \mathcal{S} \) is the \( i \)-th state and \( Q_i \in \{t, \ldots, t + n\} \) is the \( i \)-th statement index. For each \( i \geq 0 \):

(a) If \( Q_i = t + n \), then \( e_P^g(S) = \text{true} \) and \( e_P(S) = S_i \). (This terminates the execution of \( P \) normally.)

(b) Otherwise, since \( Q_i \in \{t, \ldots, t + n\} \), we must have \( t \leq Q_i \leq t + n - 1 \). If \( e_k^g(S_i) = \text{false} \) (where \( k = Q_i \), so that \( P_k \) is the current statement), then \( e_P^g(S) = \text{false} \). (We say that \( P \) encounters an unchecked error, in this case; the new current state is undefined, as, for example, with an array index out-of-bounds condition.)

(c) Otherwise, if \( e_k^g(S_i) = \text{false} \) (where \( k = Q_i \) as above), then \( e_P^g(S) = \text{false} \). (We say also, in this case, that \( P \) encounters an unchecked error; the next statement to be executed is undefined, as, for example, with a switch value out-of-bounds condition.)

(d) Otherwise — that is, if \( e_k^g(S_i) = c_k^g(S_i) = \text{true} \), where \( k = Q_i \) as above — then \( S_{i+1} = e_k(S_i) \) and \( Q_{i+1} = c_k(S_i) \) (in \( \{t, \ldots, t + n\} \) by Definition 5.1(a)). (This continues the execution of \( P \).)
Here $S_{i+1}$ and $Q_{i+1}$ are defined only in case (d) above. If the $S_i$ are hereby defined for all $i, 0 \leq i < \infty$, then $e^o_P(S) = \text{false}$, and we say that $P$ loops endlessly when started at $S$, in this case.

As opposed to the unchecked errors introduced above, a checked error would presumably involve an error exit; and this is still normal termination, in our terminology. Of course, a real computer keeps going when a run-time error is unchecked, but in an undefined manner; mathematically, the execution sequence has terminated (abnormally). Note that, if $P_k$ is the last statement of $P$, where $k = Q_i$, the execution sequence ends at $(S_{i+1}, Q_{i+1})$, not at $(S_i, Q_i)$. In particular, $e_P(S)$, in this case, is $e_k(S)$, not $S_i$, since the last statement of $P$ remains to be executed.

We shall always assume that the directed graph $G$ of $P$ is rooted at $t$, so that any vertex in $G$ is reachable from $t$. It is well known that any vertex in a graph, not reachable from its root, may be removed from the graph without affecting any paths in the graph from the root. Here, if $(S_0, Q_0) = (S, t)$, $(S_1, Q_1)$, $(S_2, Q_2)$, ..., is an execution sequence in $P$, then $Q_0 = t, Q_1, Q_2, ...$ is a path in $G$ by Definitions 5.1(c) and 5.2(d). Therefore, any vertex not reachable from $t$ cannot appear in such a sequence, and cannot, by Definition 5.2(a), change the effect of $P$ in any way.

We now come to the theorems which we discussed in the Overview (Section 1 above). It is well known that $e_P$, here, might not be computable. We might not know whether $P$ halts at all (this is the famous halting problem; see [4]); and, even if $P$ does halt under certain circumstances, determining what these circumstances are is an unsolvable problem. Nevertheless, even in this case, it is still possible to bound the conditional regions of the effect of $P$ in terms of those of the effects of its parts, as follows.

**Theorem 5.1** Using the terminology of Definition 5.2, we have \(\text{COR}(e_P, e_P^o) \subseteq \bigcup_{i=t}^{t+n-1} \text{COR}(e_i, e_i^o)\).

**PROOF.** Let $x \notin \bigcup_{i=t}^{t+n-1} \text{COR}(e_i, e_i^o)$, so that $x \notin \text{COR}(e_i, e_i^o)$ for each $i, t \leq i \leq t + n - 1$. We wish to show that $x \notin \text{COR}(e_P, e_P^o)$, so that, by Definition 2.10, we must have $e_P(S)(x) = S(x)$ for every $S \in \mathcal{S}$ such that $e_P^o(S) = \text{true}$. Accordingly, let $e_P^o(S) = \text{true}$; for this particular $S \in \mathcal{S}$, we construct the execution sequence $(S_0, Q_0) = (S, t)$, $(S_1, Q_1)$, $(S_2, Q_2)$, ..., as in Definition 5.2. This sequence must terminate at some $(S_z, Q_z)$, for $z \geq 0$; otherwise (see the end of Definition 5.2), we would have $e_P^o(S) = \text{false}$, a contradiction. We now show by induction on $j$ that $S_j(x) = S_0(x)$ (or $S(x)$) for all $j, 0 \leq j \leq z$. This is clear for $j = 0$; in general, suppose that $S_j(x) = S_0(x)$, that $S_{j+1}$ is defined, and that $k = Q_j$ as in Definition 5.2. By this definition, the conditions of part (d) must then be satisfied, so that, in particular,
In section 5.4 of [9] we showed the surprising fact that the input region of \( f \neq l \) (that is, if \( f \neq I \)) can include the output region of \( I \). The same is true of sequential programs, and for the same reason. (In the following example, we use the terminology of Definition 5.1(c) and (d).)

**Example 5.1** Consider a program \( P \) with two statements, each with no guards. Statement 1 is a simple 2-way branch, which branches to statement 3, one beyond the end (in other words, it exits) if \( S(x) \geq 0 \), and branches to statement 2 otherwise. Statement 2 is \( x \leftarrow 0 \), and it always branches to statement 1. If \( S(x) \geq 0 \), then the execution sequence starting with \((S,1)\) is \((S,1),(S,3)\), and \( e_P(S) = S \). If \( S(x) < 0 \), and \( S' \) is defined by \( S'(x) = 0 \) and \( S'(y) = S(y) \) for all \( y \in M \neq x \), then this execution sequence becomes \((S,1),(S,2),(S',1),(S',3)\), and \( e_P(S) = S' \). In particular, \( e_P(S)(x) = S'(x) \neq S(x) \), so that \( x \in \text{OR}(e_P) \). If \( S_1(x) = 1 \), \( S_2(x) = 2 \), and \( S_1(y) = S_2(y) \) for all \( y \in M \neq x \), then \( e_P(S_1)(x) = S_1(x) \neq S_2(x) = e_P(S_2)(x) \); since \( x \in \text{OR}(e_P) \), this shows that also \( x \in \text{IR}(e_P) \).

We can, however, show that the input region of \( e_P \) cannot include any more than the relevant regions of the various \( c_i \) and the input and output regions of the various \( e_i \) (all these regions being conditional).

**Theorem 5.2** Using the terminology of Definition 5.2, we have \( \text{CIR}(e_P,e_P^0) \subseteq \bigcup_{i=t}^{t+n-1} (\text{COR}(e_i,e_i^0) \cup \text{CIR}(e_i,e_i^0) \cup \text{CRR}(c_i,c_i^0)) \).

**Proof.** For each \( i,t \leq i \leq t + n - 1 \), let \( x \notin \text{COR}(e_i,e_i^0) \) (so that \( x \notin \text{COR}(e_P,e_P^0) \), by Theorem 5.1); let \( x \notin \text{CIR}(e_i,e_i^0) \) (so that \( x \notin \text{RR}(e_i^0) \), since \( \text{RR}(e_i^0) \subseteq \text{CIR}(e_i,e_i^0) \) by Theorem 2.5); and let \( x \notin \text{CRR}(c_i,c_i^0) \) (so that \( x \notin \text{RR}(e_i^0) \), since \( \text{RR}(e_i^0) \subseteq \text{CRR}(c_i,c_i^0) \) by Theorem 3.1). Now let \( S \) and \( S' \in \mathbb{S} \) be such that \( S(y) = S'(y) \) for all \( y \neq x \) and \( e_P^0(S) = \text{true} \). We wish to show that \( x \notin \text{CIR}(e,e^0) \); by Definition 2.12, it suffices to show that \( e_P^0(S') = \text{true} \) and \( e_P(S)(y) = e_P(S')(y) \) for all \( y \in \text{COR}(e_P,e_P^0) \). We construct two execution sequences, \((S_0,Q_0) = (S,t), (S_1,Q_1), (S_2,Q_2), \ldots \) from \( S \) and \((S'_0,Q'_0) = (S',t), (S'_1,Q'_1), (S'_2,Q'_2), \ldots \) from \( S' \), as in Definition 5.2. The sequence \((S_i,Q_i) \) must terminate at some \((S_z,Q_z) = t + n \); otherwise (see the end of Definition 5.2), we would have \( e_P^0(S) = \text{false} \), a contradiction. For each \( i,0 \leq i \leq z \), we now show, by induction on \( i \), that:
• $S'_i$ is defined.
• $Q'_i$ is defined, and equal to $Q_i$.
• $S'_i(y) = S_i(y)$ for all $y \in M$ ≠ $x$.

For $i = 0$, we have that:

• $S'_0$ is defined, since $S'_0 = S'$.
• $Q'_0$ is defined, and equal to $Q_0$, since $Q'_0 = t = Q_0$.
• $S'_0(y) = S'(y) = S(y) = S_0(y)$ for all $y \in M$ ≠ $x$.

In the induction step, we set $i = j$, so that:

(A) $S'_j$ is defined.
(B) $Q'_j$ is defined, and equal to $Q_j$.
(C) $S'_j(y) = S_j(y)$ for all $y \in M$ ≠ $x$.

We now show (A) through (C) above for $i = j + 1$, where $j + 1 ≤ z$, so that $j ≤ z - 1$, and thus $S_j$ and $S_{j+1}$ are both defined. Therefore the conditions of Definition 5.2(d) must hold, so that, setting $k = Q'_j (= Q_j)$, we have $t ≤ k ≤ t + n - 1$; $e'_k(S_j) = e''_k(S_j) = true; S_{j+1} = e_k(S_j)$; and $Q_{j+1} = c_k(S_j)$.

By Theorem 2.1, since $x \notin RR(e'_k)$, we have $e''_k(S'_j) = e''_k(S_j) = true$, while since $x \notin RR(c'_k)$, we have $e''_k(S'_j) = e''_k(S_j) = true$. Since $x \notin \text{CIR}(c'_i, c''_i)$, we now have $c_k(S'_j) = c_k(S_j)$ (by Theorem 3.1). Thus:

(A') $S'_{j+1} = e_k(S'_j)$ (by Definition 5.2(d)), so that, in particular, $S'_{j+1}$ is defined.
(B') $Q'_{j+1} = c_k(S'_j)$ (by Definition 5.2(d)) = $c_k(S_j) = Q_{j+1}$, so that, in particular, $Q'_{j+1}$ is defined, and equal to $Q_{j+1}$.
(C') If $y \in \text{COR}(e_k, e'_k)$, then, since $e''_k(S'_j) = e''_k(S_j) = true$, we have $e_k(S'_j)(y) = e_k(S_j)(y)$ by Definition 2.12, since $x \notin \text{CIR}(e_k, e'_k)$, and since $S'_j(y) = S_j(y)$ for all $y ≠ x$ by (C) above. If $y \notin \text{COR}(e_k, e'_k)$ but $y ≠ x$, we have $e_k(S'_j)(y) = S'_j(y)$ (by Definition 2.10, since $e''_k(S'_j) = true$) = $S_j(y)$ (since $y ≠ x$) = $e_k(S_j)(y)$ (again by Definition 2.10, since $e''_k(S_j) = true$). Hence, in all cases, $S'_{j+1}(y) = e_k(S'_j)(y) = e_k(S_j)(y) = S_{j+1}(y)$ for all $y \in M$ ≠ $x$.

Setting $i = z$, we now have that $S'_z$ is defined; $Q'_z$ is defined, and equal to $Q_z$; and $S'_z(y) = S_z(y)$ for all $y \in M$ ≠ $x$. Since $S_{z+1}$ is not defined, the conditions of Definition 5.2(d) cannot hold; since $e''_z(S) = true$, the conditions of Definition 5.2(b) and (c) cannot hold, either. Therefore the conditions of Definition 5.2(a) must hold, and $Q'_z = Q_z = t + n$. Thus these conditions hold for both $S$ and $S'$, and hence $e''_p(S') = true; e_p(S) = S_z; and e''_p(S') = S'_z$. Hence, for all $y ≠ x$ (and thus for all $y \in \text{COR}(e_p, e''_p)$, since $x \notin \text{COR}(e_p, e''_p)$), we have $e_p(S)(y) = S_z(y) = S'_z(y) = e''_p(S)(y)$. □
5.1 Simple Sections and Contraction

A contiguous single-entry, single-exit section of the statements of a sequential program is easy to work with as if it were a single statement. We define such sections as being simple, as follows.

**Definition 5.3** Given a sequential program $P$ with start index $t$ and length $n$, a simple section $\Sigma$ of $P$ is the set of statements $P_\theta, P_{\theta+1}, \ldots, P_{\theta+\lambda-1}$, for some $\theta$, $t \leq \theta \leq \theta + \lambda - 1 \leq t + n - 1$, such that:

(a) For all $i$, $t \leq i \leq \theta - 1$ and $\theta + \lambda \leq i \leq t + n - 1$, if $c_i^\theta(S) = true$, then either $c_i(S) \leq \theta$ or $c_i(S) \geq \theta + \lambda$. (This specifies $\Sigma$ as being single-entry; the next statement after any statement not in $\Sigma$ must either be not in $\Sigma$, or it must be the first statement of $P'$, which is the entry $P_\theta$ of $\Sigma$.)

(b) For all $i$, $\theta \leq i \leq \theta + \lambda - 1$, if $c_i^\theta(S) = true$, then $\theta \leq c_i(S) \leq \theta + \lambda$. (This specifies $\Sigma$ as being single-exit; the next statement after any statement of $\Sigma$ must either be in $\Sigma$ or one beyond the end of $\Sigma$, which is the exit $P_{\theta+\lambda}$ of $\Sigma$.)

If $\Sigma'$ is another simple section of $P$, then $\Sigma'$ is adjacent to $\Sigma$ if the start index of $\Sigma'$ is $\theta + \lambda$.

Strictly speaking, this definition does not preclude simple sections with no entries at all. However, a simple section of $P$ will always have at least one entry under our assumption that the directed graph of $P$ is rooted at $t$.

Among the simple sections are the basic blocks [1]. Like a simple section, a basic block is a single-entry, single-exit section; but there are further restrictions on a basic block. These are either that it contains no branches at all, or, more generally, that “each instruction...always executes before all those in later positions” [6], thus effectively forbidding if statements within a basic block. Whichever definition is used, a basic block cannot contain loops, whereas a simple section may contain loops.

**Lemma 5.1** If $\Sigma$ satisfies the conditions of Definition 5.3, it is a sequential program in its own right, having start index $\theta$ and length $\lambda$.

**Proof.** That $\Sigma$ has start index $\theta$ and length $\lambda$ is obvious from the definition. What needs to be checked is that the controller of any statement $P_i \in \Sigma$ is a function $c_i : S \rightarrow \{\theta, \ldots, \theta + \lambda\}$; that is, that $\theta \leq c_i(S) \leq \theta + \lambda$ for all $S \in S$ for which $c_i^\theta(S) = true$. But this follows from the fact that $\Sigma$ is single-exit; indeed, it is precisely Definition 5.3(b). □
We may now collapse $\Sigma$ into a single statement of a contraction $P'$ of $P$, whose statements are those of $P$ outside of $\Sigma$, together with a single sequential statement (see Definition 5.1(b)) which represents $\Sigma$ and whose effect is the overall effect of $\Sigma$. In order to do this, we first define a function $\kappa$ which specifies the statement $P_{\kappa(i)}$ in $P$ corresponding to a given statement $P_i'$ in $P'$.

**Definition 5.4** Given $t$, $n$, $\theta$, and $\lambda$ as in Definition 5.3, the correspondence function $\kappa$ is defined by $\kappa(i) = i$ for all $i$, $t \leq i \leq \theta$, and $\kappa(i) = i + \lambda - 1$ for all $i$, $\theta + 1 \leq i \leq t + n + 1 - \lambda$.

We will need eight basic properties of $\kappa$.

**Lemma 5.2** With $\kappa$ as in Definition 5.4, we have:

(a) $\kappa(t) = t$. (The first statement of $P'$ corresponds to the first statement of $P$.)

(b) $\kappa(t + n + 1 - \lambda) = t + n$. (The index one beyond the end of $P'$ corresponds to the index one beyond the end of $P$.)

(c) $\kappa(\theta) = \theta$. (The statement that represents $\Sigma$ within $P'$ corresponds to the first statement of $\Sigma$ in $P$.)

(d) $\kappa(\theta + 1) = \theta + \lambda$. (The statement in $P'$ which follows $\Sigma$ corresponds to the statement just beyond $\Sigma$ — which has length $\lambda$ — in $P$.)

(e) If $\kappa(i) = \kappa(j)$, then $i = j$. (That is, $\kappa$ is invertible.)

(f) The domain of $\kappa$ (and therefore the range of $\kappa^{-1}$) is the set of all integers $i$ such that $t \leq i \leq t + n + 1 - \lambda$. The range of $\kappa$ (and therefore the domain of $\kappa^{-1}$) is the set of all integers $i$ such that $t \leq i \leq \theta$ or $\theta + \lambda \leq i \leq t + n$. 

(g) $c_{\kappa(i)}(S)$ is in the range of $\kappa$ whenever $c_{\kappa(i)}(S) = \text{true}$, for all $i \neq \theta$, $t \leq i \leq t + n - \lambda$. (This will be needed in the proof of Lemma 5.3 below.)

(h) If $y \leq \theta$, then $\kappa^{-1}(y) = y$; if $y \geq \theta + 1$, then $\kappa^{-1}(y) = y + 1 - \lambda$. (These will be needed in the proof of Theorem 5.4 below.)

**Proof.**

(a) This follows from Definition 5.4 by setting $i = t$.

(b) Since $\theta + \lambda - 1 \leq t + n - 1$ by Definition 5.3, we have $\theta + 1 = \theta + \lambda - 1 + (2 - \lambda) \leq t + n - 1 + (2 - \lambda) = t + n + 1 - \lambda$. Thus $\theta + 1 \leq t + n + 1 - \lambda \leq t + n + 1 - \lambda$, so that $\kappa(t + n + 1 - \lambda) = (t + n + 1 - \lambda) + \lambda - 1 = t + n$ by Definition 5.4.

(c) This follows from Definition 5.4 by setting $i = \theta$.

(d) By Definition 5.4, we have $\kappa(i) = i + \lambda - 1$ for all $i$, $\theta + 1 \leq i \leq t + n + 1 - \lambda$; setting $i = \theta + 1$ here, we have $\kappa(\theta + 1) = \theta + 1 + \lambda - 1 = \theta + \lambda$.

(e) If $t \leq i \leq \theta$ and $t \leq j \leq \theta$, then $i = \kappa(i) = \kappa(j) = j$. If $\theta + 1 \leq i \leq t + n + 1 - \lambda$ and $\theta + 1 \leq j \leq t + n + 1 - \lambda$, then $i + \lambda - 1 = \kappa(i) = \kappa(j) = j + \lambda - 1$, so that again $i = j$. Otherwise, by change of variable if necessary, we may assume that $t \leq i \leq \theta$ and $\theta + 1 \leq j \leq t + n + 1 - \lambda$; but then $i < j$ and
thus \( \kappa(i) = i < j \leq j + n - 1 \) (since \( n > 0 \)) = \( \kappa(j) \), so this case holds vacuously.

(f) The first part of this follows directly from Definition 5.4. For the second part, we note that, if \( t \leq i \leq \theta \), then \( t \leq \kappa(i) (= i) \leq \theta \), while if

\[ \theta + 1 \leq i \leq t + n + 1 - \lambda, \text{ then } \theta + \lambda = \theta + 1 + \lambda - 1 \leq \kappa(i) (= i + \lambda - 1) \leq t + n + 1 - \lambda - 1 = t + n. \]

(g) Since \( \kappa(i) \) is in the range of \( \kappa \), we have \( t \leq \kappa(i) \leq \theta \) or \( \theta + \lambda \leq \kappa(i) \leq t + n \) (by (f) above). However, by assumption, \( i \neq \theta \), so that \( \kappa(i) \neq \kappa(\theta) = \theta \) (by (c) above); and \( i \leq t + n - \lambda \), implying that \( i \neq t + n + 1 - \lambda \), so that \( \kappa(i) \neq \kappa(t + n + 1 - \lambda) = t + n \) (by (b) above). Therefore, in fact, \( t \leq \kappa(i) \leq \theta - 1 \) or \( \theta + \lambda \leq \kappa(i) \leq t + n - 1 \). By Definition 5.3(a), therefore (in other words, since \( \Sigma \) is single-entry), since \( c_{\kappa(i)}(\Sigma) = \text{true} \) by hypothesis, we have \( c_{\kappa(i)}(\Sigma) \leq \theta \) or \( c_{\kappa(i)}(\Sigma) \geq \theta + \lambda \). Since \( c_{\kappa(i)} : \mathfrak{S} \to \{t, \ldots, t + n\} \) by Definition 5.1(a), we have \( t \leq \kappa(i) \leq \theta \) or \( \theta + \lambda \leq \kappa(i) \leq t + n \), so that \( c_{\kappa(i)}(\Sigma) \) is in the range of \( \kappa \) by (f) above.

(h) Let \( x = \kappa^{-1}(y) \), so that \( \kappa(x) = y \). By Definition 5.4, either \( y = x \) or \( y = x + \lambda - 1 \). Suppose first that \( y \leq \theta \). If \( y = x + \lambda - 1 \), then \( \theta + 1 \leq x \leq t + n + 1 - \lambda \), and thus \( \theta + 1 \leq \theta + \lambda = \theta + 1 + \lambda - 1 \leq x + \lambda - 1 = y \), a contradiction. Hence \( y = x \), and \( \kappa^{-1}(y) = x = y \). Now suppose that \( y \geq \theta + 1 \). If \( y = x \), then \( t \leq x \leq \theta \), a contradiction. Hence \( y = x + \lambda - 1 \), and \( \kappa^{-1}(y) = x = y + 1 - \lambda \). \( \square \)

Having defined \( \kappa \), we can now define the contraction \( P' \).

**Definition 5.5** Given \( P, \Sigma, t, n, \theta, \) and \( \lambda \) as in Definition 5.3, the **contraction** of \( P \) with contracted section \( \Sigma \) is a sequential program \( P' \) with start index \( t \) and length \( n + 1 - \lambda \), defined as follows. The statements of \( P' \) are \( P_1, P_{t+1}, \ldots, P_{\theta}, P_{\theta+\lambda}, P_{\theta+\lambda+1}, \ldots, P_{t+n-1} \); they may also be denoted by \( P'_1, \ldots, P'_{t+n-\lambda} \). Here \( P'_i = P_{\kappa(i)} \) for all \( i \), \( t \leq i \leq t + n + 1 - \lambda \), where \( \kappa \) is as in Definition 5.4; and \( P'_\theta \) (sometimes called \( P'_\Sigma \); see the proof of Theorem 6.1 below) is the **contracted statement**, the result of contracting \( \Sigma \) to one statement of \( P' \). The effect \( e'_i \) and the controller \( c'_i \) of each statement \( P'_i \) of \( P' \) are defined as:

(a) \( (e'_1, e'_{\theta}) = (e_\Sigma, e'_\Sigma) \). (The effect of the contracted statement is the overall effect of the section \( \Sigma \) as a sequential program, as in Definition 5.2.)

(b) \( (e'_S, e''_S) \equiv (\theta + 1, \text{true}) \), for all \( S \in \mathfrak{S} \). (The contracted statement is a sequential statement, as this is specified in Definition 5.1(b).)

Then, for all \( i \neq \theta \), \( t \leq i \leq t + n - \lambda \), we have:

(c) \( (e'_i, e''_i) = (e_{\kappa(i)}, e''_{\kappa(i)}) \). (The effect of any statement \( P'_i \) in \( P' \), other than the contracted statement, is the same, and under the same conditions, as the effect of the corresponding statement \( P_{\kappa(i)} \) in \( P \).)

(d) \( c'_i = c''_{\kappa(i)} \). (The controller of statement \( P'_i \) in \( P' \) is defined under the same conditions as is the controller of the corresponding statement \( P_{\kappa(i)} \) in \( P \).)
Lemma 5.3  The specifications of Definition 5.5 actually define $P'$ as a sequential program.

PROOF. There are two points to verify, both of which have to do with part (e). The first is that $c_{\kappa(i)}(S)$ is actually in the domain of $\kappa^{-1}$, or the range of $\kappa$, so that $c_i'(S)$ is properly defined; but this holds by Lemma 5.2(g). The second is that the values of $c_i'(S)$ are all in their proper range. But $c_{\kappa(i)}(S)$ is in the domain of $\kappa^{-1}$, and thus $\kappa^{-1}(c_{\kappa(i)}(S))$ is in the range of $\kappa^{-1}$, or the domain of $\kappa$; that is, $t \leq c_i'(S) = \kappa^{-1}(c_{\kappa(i)}(S)) \leq t + n + 1 - \lambda$, by Lemma 5.2(f). This is the proper range, by Definition 5.1(a), since $P'$ has start index $t$ and length $n + 1 - \lambda$. □

We emphasize the fact that Definition 5.5 above works only for single-entry, single-exit sections. It relies on the fact that a simple section $\Sigma$ is a sequential program in its own right, which in turn depends on $\Sigma$ being single-exit; also, Definition 5.5(e), as we have seen, only works if $\Sigma$ is single-entry. This appears to account for the unusual length of the proof of the following theorem, whose conclusion might appear obvious; namely, that the process of contraction of a sequential program leaves its effect unchanged.

Theorem 5.3  If $P'$ is the contraction of $P$ with contracted section $\Sigma$, in Definition 5.5, then $(e_P', e_{P'}^g) = (e_P, e_P^g)$.

PROOF. We assume that every statement $P_i$ in $P$ has an effect $e_i$ and a controller $c_i$, and that every statement $P_i'$ in $P'$ has an effect $e_i'$ and a controller $c_i'$. The respective overall effects of $P$ and of $P'$ are denoted here by $e_P$ and $e_{P'}$ (not $e_{P'}'$). As in Definition 5.5, we assume that $P$ has start index $t$ and length $n$, and that $P'$ has start index $t$ and length $n + 1 - \lambda$. For each $S \in \mathbb{S}$, we need to show, by Definition 2.14, that either

$$e_{P'}^g(S) = e_P^g(S) = false$$

or else that

$$e_{P'}^g(S) = e_P^g(S) = true \quad \text{and} \quad e_{P'}(S) = e_P(S)$$

Let $S \in \mathbb{S}$. We construct the execution sequences $(S_i, Q_i)$ of $P$ and $(S_i', Q_i')$ of $P'$, starting at $(S, t)$, together with a function $\gamma$ (where $\gamma(0) = 0$) such that
\((S'_i, Q'_i)\) corresponds to \((S_{\gamma(i)}, Q_{\gamma(i)})\); that is:

- \(S'_i = S_{\gamma(i)}\). (Corresponding states are the same in the two sequences.)
- \(\kappa(Q'_i) = Q_{\gamma(i)}\). (Given the statement index \(Q'_i\) in \(P'\), and applying to it the function \(\kappa\) as in Definition 5.4 to obtain the corresponding index in \(P\), this is \(Q_{\gamma(i)}\), with the sequence index \(\gamma(i)\) corresponding to \(i\).)
- \(\gamma(i) \geq i\). (The sequence in \(P\) may contain subsequences in \(\Sigma\), each of which will have been replaced by a single element of the sequence in \(P'\).)

This is true for \(i = 0\), since:

- \(S'_0 = S = S_0 = S_{\gamma(0)}\).
- \(\kappa(Q'_0) = \kappa(t) = t\) (by Lemma 5.2(a)) = \(Q_0 = Q_{\gamma(0)}\).
- \(\gamma(0) = 0 \geq 0\).

We assume this to hold for \(i = j\), that is:

\[\begin{align*}
(A) \quad & S'_j = S_{\gamma(j)} \\
(B) \quad & \kappa(Q'_j) = Q_{\gamma(j)} \\
(C) \quad & \gamma(j) \geq j
\end{align*}\]

and prove it for \(i = j + 1\), simultaneously constructing \(\gamma(j + 1)\). That is, we need to prove:

\[\begin{align*}
(A') \quad & S'_{j+1} = S_{\gamma(j+1)} \\
(B') \quad & \kappa(Q'_{j+1}) = Q_{\gamma(j+1)} \\
(C') \quad & \gamma(j + 1) \geq j + 1
\end{align*}\]

We shall prove \((A')\), \((B')\), and \((C')\) above whenever \(S'_{j+1}\) and \(Q'_{j+1}\) are actually defined. Of course, these will not be defined when \((S'_j, Q'_j)\) is the last element of the execution sequence of \(P'\). If this is a normally terminating execution sequence, then we will have

\[Q'_j = t + n + 1 - \lambda\]  \hspace{1cm} (3)

by Definition 5.2(a). This will be true if and only if \(\kappa(Q'_j) = \kappa(t + n + 1 - \lambda)\), by Lemma 5.2(e). In turn, this is true if and only if \(Q_{\gamma(j)} = \kappa(Q'_j)\) (by \((B)\) above) = \(\kappa(t + n + 1 - \lambda) = t + n\) (by Lemma 5.2(b)); in other words, if and
only if
\[ Q_{\gamma(j)} = t + n \] (4)

which implies, by Definition 5.2(a), that \((S_{\gamma(j)}, Q_{\gamma(j)})\) is the last element of the normally terminating execution sequence \((S_i, Q_i)\) of \(P\). Thus (3) and (4) above are either both true or both false. If they are both true, then \(S'_{j+1}\) and \(Q'_{j+1}\) are not defined, and we postpone consideration of this case until the end of the proof.

We assume now, therefore, that (3) and (4) above are both false; that is,
\[ Q_{\gamma(j)} \neq t + n \quad \text{and} \quad Q'_j \neq t + n + 1 - \lambda \] (5)

Thus neither \((S_{\gamma(j)}, Q_{\gamma(j)})\) nor \((S'_j, Q'_j)\) is the last element of its execution sequence. By Definition 5.2, we have \(Q_{\gamma(j)} \in \{t, \ldots, t + n\}\), and also \(Q'_j \in \{t, \ldots, t + n + 1 - \lambda\}\) (since \(P'\) has start index \(t\) and length \(n + 1 - \lambda\) by Definition 5.5). It follows from this that \(t \leq Q_{\gamma(j)} \leq t + n - 1\) if and only if
\[ t \leq Q'_j \leq t + n + 1 - \lambda - 1 = t + n - \lambda \] (6)

We set \(w' = Q'_j\) and \(w = \kappa(w')\), so that \(w\) is the statement index in \(P\) corresponding to the statement index \(w'\) in \(P'\). We thus have \(w = \kappa(w') = \kappa(Q'_j)\) (since \(w' = Q'_j\) = \(Q_{\gamma(j)}\) (by (B) above); that is,
\[ w = Q_{\gamma(j)} \] (7)

so that \(w\) is also the statement index of element \(\gamma(j)\) of the execution sequence of \(P\), corresponding to the index of element \(j\) of the execution sequence of \(P'\), which is \(Q'_j\), or \(w'\). Setting \(i = w'\) in Definition 5.5, and noting that \(\kappa(i) = \kappa(w') = w\), we have \(e_{w'}^{tg}(S'_{j}) = e_{w}^{tg}(S_{\gamma(j)})\) by Definition 5.5(c) and \(c_{w'}^{tg}(S'_{j}) = c_{w}^{tg}(S_{\gamma(j)})\) by Definition 5.5(d). Since \(S'_{j} = S_{\gamma(j)}\) by (A) above, we therefore have
\[ e_{w'}^{tg}(S'_{j}) = e_{w}^{tg}(S_{\gamma(j)}) \] (8)
\[ c_{w'}^{tg}(S'_{j}) = c_{w}^{tg}(S_{\gamma(j)}) \] (9)

We now take up the case in which \(S'_{j+1}\) and \(Q'_{j+1}\) are not defined because \((S'_j, Q'_j)\) is the last element of an abnormally terminating execution sequence of \(P'\). This will hold for either of two reasons:

- \(e_{w'}^{tg}(S'_{j}) = false\) (Definition 5.2(b)); this implies, by (8) above, that \(e_{w}^{tg}(S_{\gamma(j)}) = false\). Applying this definition to \(P\) and setting \(i = \gamma(j)\), we have \(k =
\[ Q_{\gamma(j)} = w, \text{ so that, if } e_{w}(S_{\gamma(j)}) = false, \text{ then } e_{p}(S) = false. \] Also, applying the definition to \( P' \) and setting \( i = j \), we have \( k = Q'_j = w' \), so that, if \( e_{w}'(S'_j) = false \), then \( e_{p'}(S) = false. \)

- \( e_{w}'(S'_j) = false \) (Definition 5.2(c)); this implies, by (9) above, that \( e_{w}(S_{\gamma(j)}) = false. \) Applying this definition to \( P \) and setting \( i = \gamma(j) \), we have \( k = w \) as before, so that, if \( e_{w}(S_{\gamma(j)}) = false, \) then \( e_{p}(S) = false. \). Also, applying the definition to \( P' \) and setting \( i = j \), we have \( k = w' \), again as before, so that, if \( e_{w}'(S'_j) = false, \) then \( e_{p'}(S) = false. \)

Therefore, in either case, \( e_{p}(S) = false = e_{p'}(S) \), which proves (1) above; and we are done, in these cases. Hence we may assume that

\[ e_{w}'(S'_j) = e_{w}'(S'_j) = e_{w}(S_{\gamma(j)}) = true \]  

(10)

so that \( e_{w}'(S'_j), e_{w}'(S'_j), e_{w}(S_{\gamma(j)}), \) and \( c_{w}(S_{\gamma(j)}) \) are all defined, and the conditions of Definition 5.2(d) hold. Applying this definition to \( P \) and setting \( i = \gamma(j) \), so that \( k = Q_i = Q_{\gamma(j)} = w \) (by (7) above), we have

\[ S_{\gamma(j)+1} = e_{w}(S_{\gamma(j)}) \]  

(11)
\[ Q_{\gamma(j)+1} = c_{w}(S_{\gamma(j)}) \]  

(12)

Also, applying this definition to \( P' \) and setting \( i = j \), so that \( k = Q'_i = Q'_j = w' \) as above, we have

\[ S'_{j+1} = e_{w}'(S'_j) \]  

(13)
\[ Q'_{j+1} = c_{w}'(S'_j) \]  

(14)

We now have two cases. If \( Q'_j \neq \theta \), then we extend the function \( \gamma \) by setting

\[ \gamma(j + 1) = \gamma(j) + 1 \]  

(15)

Applying Definition 5.5(c), and setting \( S = S'_j \) and \( i = w'(= Q'_j \neq \theta) \), so that \( k(i) = k(w') = w \), and since \( t \leq w' = Q'_j \leq t + n - \lambda \), by (6) above, we have

\[ e_{w}'(S'_j) = e_{w}(S'_j) \]  

(16)

since \( e_{w}'(S'_j) = true \) by (10) above. Then, applying Definition 5.5(e), and, as before, setting \( S = S'_j \) and \( i = w' \), so that \( k(i) = w, i \neq \theta, \) and \( t \leq w' \leq t + n - \lambda \), we have

\[ c_{w}'(S'_j) = \kappa^{-1}(c_{w}(S'_j)) \]  

(17)
since \( c'_w(S'_j) = \text{true} \) by (10) above. We now prove (A'), (B'), and (C') above as follows:

\( (A') \ S'_{j+1} = e'_w(S'_j) \) (by (13) above)  
\[ = e_w(S'_j) \] (by (16) above)  
\[ = e_w(S_{\gamma(j)}) \] (since \( S'_j = S_{\gamma(j)} \), by (A) above)  
\[ = S_{\gamma(j)+1} \] (by (11) above)  
\[ = S_{\gamma(j+1)} \] (by (15) above)

\( (B') \ \kappa(Q'_{j+1}) = \kappa(c'_w(S'_j)) \) (by (14) above)  
\[ = \kappa(\kappa^{-1}(c_w(S'_j))) \] (by (17) above)  
\[ = c_w(S'_j) \] (by the definition of the inverse)  
\[ = c_w(S_{\gamma(j)}) \] (since \( S'_j = S_{\gamma(j)} \), by (A) above)  
\[ = Q_{\gamma(j)+1} \] (by (12) above)  
\[ = Q_{\gamma(j+1)} \] (by (15) above)

\( (C') \ \gamma(j+1) = \gamma(j) + 1 \) (by (15) above)  
\[ \geq j + 1 \] (by (C) above)

In our second case, when \( Q'_j = \theta \), we construct, as in Definition 5.2, an execution sequence \( (S''_i, Q''_i) = (S'_i, Q'_i) = (S'_j, \theta), (S''_1, Q''_1), (S''_2, Q''_2), \ldots \) for \( \Sigma \). For every \( i \) for which \( (S''_i, Q''_i) \) is defined, we have

\[ (S''_i, Q''_i) = (S_{i+\gamma(j)}, Q_{i+\gamma(j)}) \] \hspace{1cm} (18)

as may be shown by induction on \( i \). It holds for \( i = 0 \), since \( (S''_0, Q''_0) = (S'_j, \theta) \); \( S'_j = S_{\gamma(j)} \) (by (A) above) \( = S_{0+\gamma(j)} \); and \( \theta = \kappa(\theta) \) (by Lemma 5.2(c)) \( = \kappa(Q'_j) \) \( = Q_{\gamma(j)} \) (by (B) above) \( = Q_{0+\gamma(j)} \). For the inductive step, suppose that \( (S''_i, Q''_i) = (S_{i+\gamma(j)}, Q_{i+\gamma(j)}) \), and that \( (S''_{i+1}, Q''_{i+1}) \) is defined, so that the conditions of Definition 5.2(d) hold for \( \Sigma \). Setting \( k = Q'_i \) \( (= Q_{i+\gamma(j)}) \), we have \( S''_{i+1} = e_k(S''_i) = e_k(S_{i+\gamma(j)}) = S_{i+1+\gamma(j)} \) and \( Q''_{i+1} = c_k(S''_i) = c_k(S_{i+\gamma(j)}) = Q_{i+1+\gamma(j)} \).

Thus the sequence \( (S''_i, Q''_i) \) for \( \Sigma \) is a part of the sequence \( (S_i, Q_i) \) for \( P \). There are now three subcases.

(i) If the \( (S''_i, Q''_i) \) are defined for all \( i \), \( 0 \leq i < \infty \), then \( \Sigma \) enters an endless loop. In this case, by (18) above, the \( (S_i, Q_i) \) are also defined for all \( i \), \( 0 \leq i < \infty \), so that \( P \) also enters an endless loop. Hence \( c'_w(S'_j) = \text{false} \), and
\( e_\theta^g(S) = \text{false} \) (see the end of Definition 5.2), so that \( e_\theta^g(S'_i) = e_\Sigma^g(S'_i) \) (by Definition 5.5(a)) = \( \text{false} \). Applying Definition 5.2(b) to \( P' \), and setting \( i = j \), so that \( k = Q'_i = Q'_j = \theta \), we obtain that, if \( e_\theta^g(S'_i) = \text{false} \) (which we have just shown), then \( e_\theta^g(S) = \text{false} = e_\Sigma^g(S) \) (as above). This proves (1) above, and we are done in this case.

\((ii)\) Now suppose that the \((S'_i, Q'_i)\) are defined only for \( 0 \leq i \leq z \). Here we must have

\[ z > 0 \quad \text{(19)} \]

(since, if \( z = 0 \), then \((S'_0, Q'_0) = (S''_0, \theta + \lambda) \) by Definition 5.2(a); but this is impossible, since \((S'_0, Q'_0) = (S'_j, \theta) \) by definition, and \( \theta \neq \theta + \lambda \) since \( \lambda > 0 \)). In this case, by (18) above, the \((S_i, Q_i)\) are defined for \( 0 \leq i \leq z + \gamma(j) \). Setting \( w'' = Q''_z = Q_{z+\gamma(j)} \), if \( e_{w''}^g(S''_z) = \text{false} \), or if \( e_{w''}^g(S''_z) = \text{false} \), then \( \Sigma \) terminates abnormally. Also, either \( e_{w''}^g(S_{z+\gamma(j)}) = e_{w''}^g(S''_z) = \text{false} \) or \( e_{w''}^g(S_{z+\gamma(j)}) = e_{w''}^g(S''_z) = \text{false} \), so that \( P \) also terminates abnormally. Thus \( e_\Sigma^g(S'_i) = \text{false} \) and \( e_\Sigma^g(S) = \text{false} \) (see Definition 5.2(b) or (c)). As before, \( e_\Sigma^g(S'_j) = e_\Sigma^g(S''_j) \) (by Definition 5.5(a)) = \( \text{false} \); and this, as before, implies that \( e_\Sigma^g(S) = \text{false} = e_\Sigma^g(S) \), and proves (1) above, and again we are done.

\((iii)\) With \( w'' \) as above, if \( e_{w''}^g(S''_z) = e_{w''}^g(S''_z) = \text{true} \), then the sequence \((S'_i, Q'_i)\) terminates normally at \((S''_z, Q''_z)\); thus the conditions of Definition 5.2(a) now hold, as applied to \( \Sigma \). Setting \( i = z \), so that \( k = Q''_z = w'' \), we now have

\[ Q''_z = \theta + \lambda \quad \text{(20)} \]

as well as \( e_\Sigma^g(S'_j) = \text{true} \) and

\[ e_\Sigma(S'_j) = S''_z \quad \text{(21)} \]

The sequence \((S''_z, Q''_z)\) has \( z + 1 \) elements, corresponding to a single element of the sequence for \( P' \); thus we extend the function \( \gamma \) by setting

\[ \gamma(j + 1) = \gamma(j) + z \quad \text{(22)} \]

in this case. By setting \( i = z \) in (18) above, we obtain

\[ (S''_z, Q''_z) = (S_{z+\gamma(j)}, Q_{z+\gamma(j)}) \quad \text{(23)} \]

since the \((S''_i, Q''_i)\) are defined for \( 0 \leq i \leq z \), and thus, in particular, for \( i = z \).

We now apply Definition 5.2 to \( P' \), setting \( i = j \), so that \( k = Q'_i = Q'_j = \theta \). By Definition 5.3, we have \( \theta + \lambda - 1 \leq t + n - 1 \), so that \( \theta \leq t + n - \lambda \), implying
that \( Q'_j = \theta \neq t + n + 1 - \lambda \); so the conditions of Definition 5.2(a) do not apply, and those of Definition 5.2(d) must apply, from which we obtain

\[
S'_{j+1} = e'_\theta(S'_j) \quad \text{(24)}
\]
\[
Q'_{j+1} = c'_\theta(S'_j) \quad \text{(25)}
\]

Finally, we have \( e'_\theta(S'_j) = e''_w(S'_j) \) (since \( w' = Q'_j = \theta \)) = \text{true} \text{ (by (10) above)}; therefore, by setting \( S = S'_j \) in Definition 5.5(a), we obtain

\[
e'_\theta(S'_j) = e_{\Sigma}(S'_j) \quad \text{(26)}
\]

We may now prove (A’), (B’), and (C’) above as follows:

(A’) \( S'_{j+1} = e'_\theta(S'_j) \) \text{ (by (24) above)}
\( = e_{\Sigma}(S'_j) \) \text{ (by (26) above)}
\( = S''_z \) \text{ (by (21) above)}
\( = S_{\gamma(j)+z} \) \text{ (by (23) above)}
\( = S_{\gamma(j+1)} \) \text{ (by (22) above)}

(B’) \( \kappa(Q'_{j+1}) = \kappa(e'_\theta(S'_j)) \) \text{ (by (25) above)}
\( = \kappa(\theta + 1) \) \text{ (by Definition 5.5(b))}
\( = \theta + \lambda \) \text{ (by Lemma 5.2(d))}
\( = Q''_z \) \text{ (by (20) above)}
\( = Q_{\gamma(j)+z} \) \text{ (by (23) above)}
\( = Q_{\gamma(j+1)} \) \text{ (by (22) above)}

(C’) \( \gamma(j + 1) = \gamma(j) + z \) \text{ (by (22) above)}
\( \geq j + z \) \text{ (by (C) above)}
\( \geq j + 1 \) \text{ (since } z > 0 \text{ by (19) above)}

Suppose now that the \( S'_i \) and the \( Q'_i \) are hereby defined for all values of \( i, 0 \leq i < \infty \). Here we have the case in which \( \Sigma \) does not enter an endless loop, but \( P' \) does; and, therefore, so does \( P \). (Otherwise, some \( (S_i, Q_i) \) would not be defined, and so neither would \( (S'_j, Q'_j) \) for all \( j \geq i \) — and, in particular, for \( j = \gamma(i) \) (since \( \gamma(i) \geq i \) by (C) above), so that \( (S'_i, Q'_i) = (S_{\gamma(i)}, Q_{\gamma(i)}) \) would
not be defined, and $P'$ would not enter an endless loop.) Therefore $e_{P'} = false$ and $e_p = false$ (see the end of Definition 5.2), proving equation (1) above.

Finally, we return to the case in which (3) and (4) above both hold. In this case, Definition 5.2(a), applied to $P$, implies that $e_{P}(S) = true$ and $e_p(S) = S_{\gamma(j)}$. Also, $n + 1 - \lambda$ is the length of $P'$, by Definition 5.5; therefore Definition 5.2(a), applied to $P'$, implies that $e_{P'}(S) = true$ and $e_{P'}(S) = S'_{\gamma(j)}$. Since $S'_{\gamma(j)} = S_{\gamma(j)}$, by (A) above, this gives $e_{P}(S) = e_{P'}(S)$, proving equation (2) above. □

5.2 A Contraction Example

Figures 1 through 5 illustrate the workings of Theorem 5.3. Figure 1 shows an original program $P$ at the left, with eight statements, $P_1$ through $P_8$; the corresponding contraction $P'$ is shown at the right, with four statements, $P'_1$ through $P'_4$. Here $P$ has a simple section $\Sigma$ consisting of $P_2$ through $P_6$, which has been contracted into the single statement $P'_2$ of $P'$. The correspondence function $\kappa$ is shown in the center of the diagram. Thus $P'_1$ corresponds to $P_1$, and so $\kappa(1) = 1$; but $P'_3$ corresponds to $P_7$, and so $\kappa(3) = 7$.

5.2.1 Program Notation

We will adopt a notation for the statements of a sequential program, as illustrated in the figure. Each statement starts with a label, which is an integer followed by a colon; and these integers are in sequence. The point one beyond the end of each program also has a label followed by a colon, although there is no statement there (here 9: in $P$, and 5: in $P'$). The correspondence function extends to one-beyond-the-end, so that $\kappa(5) = 9$, here.

Following the label of each statement is a specification of the effect, the controller, and (where necessary) the guard of that statement. The effect is given by one or more assignments, as in Definition 13 of [9], each followed by a semicolon. Thus $P'_2$ has two assignments, $a \leftarrow b \ast c$; and $d \leftarrow 0$; . The controller, if it is constant, is given by $\rightarrow$ followed by its constant value. Thus in $P_1$ we have one assignment ($u \leftarrow v$) for the effect, followed by a semicolon, followed by $\rightarrow 2$ for the controller, because this statement always proceeds to $P_2$.

If the controller is not constant, it will, in these simple examples, always have two values, say $\alpha$ and $\beta$; and there will be a condition $cond$, as in Definition 2.8, specifying whether the value of the controller is $\alpha$ (if $cond$ is true) or $\beta$ (if $cond$ is false). In this case our terminology resembles that of the question-mark operator in C, C++, or Java; it is of the form $cond? \rightarrow \alpha :\rightarrow \beta$ . Thus in $P_6$, here, we have $d = 0? \rightarrow 7 :\rightarrow 4$ ; if $d = 0$, the next statement here is $P_7$, while otherwise it is $P_4$. The effect specification is omitted in this case because
$P_6$ changes no variables, so that it has the null effect (see Definition 5.1(c)). Guards are omitted here if they are null guards (that is, always true); otherwise, they are enclosed in parentheses (see the discussion of the guard of $P'_2$ in Section 5.2.2 below).

A finite execution sequence of $P$, with the corresponding execution sequence of $P'$, is shown in Figure 2. There are seven variables in $P$ and in $P'$, namely $a$, $b$, $c$, $d$, $u$, $v$, and $w$; and each state $S_i$, or $S'_i$, is given by a 7-tuple of their values. Thus $S_0$ is given by $\{7, 2, 11, 6, 9, 1, 5\}$, signifying that $S_0(a) = 7$, $S_0(b) = 2$, $S_0(c) = 11$, $S_0(d) = 6$, $S_0(u) = 9$, $S_0(v) = 1$, and $S_0(w) = 5$. Note that $S'_0$, here, is the same as $S_0$. This execution sequence of $P$ is also shown in Figure 3, with the corresponding execution sequence of the section $\Sigma$.

An infinite execution sequence of $P$, with the corresponding execution sequence of $P'$, is illustrated in Figure 4. Note that an infinite execution sequence of $P$ could correspond to an infinite execution sequence of $P'$; but it could also, as here, correspond to an execution sequence of $P'$ which terminates abnormally. In particular, this time $b = 0$, so the guard ($b > 0$) of $P'_2$ is false, and the sequence $Q'_1$ terminates abnormally at $Q'_1$. The first few $S_i$ and $Q_i$ are given here as an indication that this sequence is infinite.

The notation used in Figure 1 may be simplified by omitting the specifications of controllers for all sequential statements, and then omitting all labels which are, or become, unused. This is shown in Figure 5 (for the left side of Figure 1 only); it is closer to notations for practical programs.

5.2.2 The Definitions and the Theorem

The program $P$ is a sequential program according to Definition 5.1, with start index $t = 1$ and length $n = 8$. To verify this, we have to check that the values of the controller of each statement in $P$ are all in the range \{t, \ldots, t + n\}, or \{1, \ldots, 9\}; but this is clear from the statement numbers following the $\rightarrow$ symbol in Figure 1.

The section from $P_2$ through $P_6$ is a simple section according to Definition 5.3, with start index $\theta = 2$ and length $\lambda = 5$. To verify this, we have to check that:

- For all $i$, $1 \leq i \leq 1$ and $7 \leq i \leq 8$, if $c_i(S) = \text{true}$, then $c_i(S) \leq 2$ or $c_i(S) \geq 7$, by Definition 5.3(a). (This affects $P_1$, $P_7$, and $P_8$, and the values of the corresponding controllers are 2, 8, and 9.)
- For all $i$, $2 \leq i \leq 6$, if $c_i(S) = \text{true}$, then $2 \leq c_i(S) \leq 7$, by Definition 5.3(b). (This affects $P_2$ through $P_6$, and the values of the corresponding controllers are 3, 4, 5, 6, 7, and again 4.)

The section $\Sigma$ in this example performs multiplication by successive addition.
Here $a$ is initialized to 0 and increased by $c$ each time through the loop; $d$ is initialized to $b$ and decreased by 1 each time through the loop; and $a$ and $d$ are the only variables which are changed by statements in $\Sigma$. Hence, by Theorem 5.1, $a$ and $d$ are the only variables which can be changed by $e_\Sigma$. In fact, $\Sigma$ sets $a$ to $b+c$, and $d$ to 0, unless $b \leq 0$, in which case $\Sigma$ loops endlessly; therefore $e_\Sigma^2$ is $b > 0$ and $e_\Sigma$ is $a \leftarrow b*c; d \leftarrow 0$. By Definition 5.5(a), therefore, the effect of the contracted section $P_2'$ is $e_\Sigma = a \leftarrow b*c; d \leftarrow 0$, and its guard is $e_\Sigma^2 = b > 0$; while by Definition 5.5(b), the controller of $P_2'$ is $\rightarrow 3 (= \theta + 1)$.

Most execution sequences resemble these in that, usually, only one variable is changed at a time. Looking at $S_0$ and $S_1$ in Figure 2, for example, we see that they are exactly the same except for the position in the $u$-column, where we have 9 for $S_0$, and 1 for $S_1$ (that is, $S_0(u) = 9$ and $S_1(u) = 1$). Sometimes no variables at all are changed; thus $S_5$ and $S_6$ are exactly the same, as are $S_8$ and $S_9$. This happens when the current statement is a conditional statement, which has the null effect. Just as there is the correspondence function $\kappa$ for statements, in the middle of Figure 1, there is the correspondence function $\gamma$ for execution sequences, in the middle of Figure 2. For example, $\gamma(4) = 11$, denoting the fact that $Q'_4$ corresponds to $Q_{11}$ (note that $S'_4$ and $S_{11}$ are the same).

The execution sequence for $P$, on the left, in Figures 2 and 4, is $(S_0, Q_0), (S_1, Q_1), \ldots$; the sequence for $P'$, on the right, is $(S'_0, Q'_0), (S'_1, Q'_1), \ldots$. The values of the $Q_i$ and the $Q'_i$ refer to the statements in Figure 1. For example, since $Q_8 = 6$ in Figure 2, this refers to $P_6$ in Figure 1, which is $d = 0? \rightarrow 7 : \rightarrow 4$. This changes no variables; and this explains why $S_8$, in Figure 2, is the same as the following state, $S_9$. Similarly, we have $Q'_1 = 2$ in Figure 2, referring to $P'_2$ in Figure 1. This changes two variables, $a$ and $d$; and this explains why $a$ and $d$ (and only $a$ and $d$) are different, in $S'_1$, from what they are in the next state, $S'_2$.

The execution sequence for $\Sigma$ in Figure 3 satisfies equation 18 in the proof of Theorem 5.3, that is, $(S''_i, Q''_i) = (S_{i+\gamma(j)}, Q_{i+\gamma(j)})$ for every $i$ for which $(S''_i, Q''_i)$ is defined. Looking again at Figure 2, we see that $\theta = 2$ and $j = 1$ (so that $Q'_j = Q'_1 = 2 = \theta$, as we have been assuming). Thus $\gamma(j) = \gamma(1) = 1$, and equation 18 becomes $(S''_i, Q''_i) = (S_{i+1}, Q_{i+1})$, which is clear from Figure 3 (thus $(S''_0, Q''_0) = (S_1, Q_1)$; $(S''_1, Q''_1) = (S_2, Q_2)$; and so on). The $(S''_i, Q''_i)$ are hence defined only for $0 \leq i \leq 8$, so that $z = 8$ in equation 19 in the proof of Theorem 5.3 (this will be used below).

Corresponding statements in $P$ and in $P'$, in Figure 1, are not always identical, because of the correspondence function $\kappa$. For example, $P'_3$, which is $w \leftarrow u + a; \rightarrow 4$, corresponds to $P_7$, which is $w \leftarrow u + a; \rightarrow 8$ (with 8 instead of 4). This illustrates the equation (B) $\kappa(Q'_j) = Q_{\gamma(j)}$ in the proof of Theorem 5.3. Setting
Fig. 1. A program with a section, and its contraction.

\[ j = 3 \] here, we have \( \kappa(4) = \kappa(Q'_3) \) (from Figure 2) = \( Q_{\gamma(3)} \) (from (B) above) = \( Q_{10} \) (from Figure 2) = 8 (again from Figure 2); and this is right, since \( \kappa(4) = 8 \) (from Figure 1).

The values of \( \kappa \) in Figure 1 are taken from Definition 5.4. Since \( t = 1, \theta = 2, n = 8, \) and \( \lambda = 5, \) we have \( \kappa(i) = i \) for all \( i, 1 \leq i \leq 2, \) and \( \kappa(i) = i + 5 - 1 \) (\( = i + 4 \)) for all \( i, 3 \leq i \leq 5. \) Thus \( \kappa(1) = 1, \kappa(2) = 2, \kappa(3) = 7, \kappa(4) = 8, \) and \( \kappa(5) = 9. \)

The values of \( \gamma \) in Figure 2 are taken from equations 15 and 22 for \( \gamma \) in the proof of Theorem 5.3. Originally, \( \gamma(0) = 0 \) (see the start of the proof). If \( j = 1, \) so that \( \gamma(j) = \gamma(1) = 1 \) and \( Q'_j = Q'_1 = 2 = \theta, \) then equation 22 applies, so that \( \gamma(2) = \gamma(j + 1) = \gamma(j) + z = 1 + z = 9 \) (since \( z = 8, \) as we saw above). In all other cases in Figure 2 (and also in Figure 4), we have \( Q'_j \neq \theta, \) so that equation 15 applies, and \( \gamma(j + 1) = \gamma(j) + 1. \) Since \( \gamma(2) = 9, \) therefore, we have \( \gamma(3) = 10, \) and then \( \gamma(4) = 11; \) and so on.

5.3 Embedded and Induced Sections

An embedded section is a section contained in another section. Such sections retain their basic properties, as follows.

**Lemma 5.4** Let \( \Sigma \) be a section of \( P, \) and let \( \Sigma' \) be a simple section of \( P, \) contained within \( \Sigma. \) Then \( \Sigma' \) is a simple section of \( \Sigma. \) (Note that \( \Sigma \) is not necessarily simple.)

**Proof.** Let \( P \) have start index \( t \) and length \( n; \) let \( \Sigma \) have start index \( \theta \) and length \( \lambda; \) and let \( \Sigma' \) have start index \( \theta' \) and length \( \lambda'. \) Since \( \Sigma' \) is contained
Also let contained within Σ, we must have to say that Σ

\[ \gamma(0) = 0 \quad Q'_0 = 1 \quad S'_0 = \{ 7 2 1 1 6 9 1 5 \} \]
\[ \gamma(1) = 1 \quad Q'_1 = 2 \quad S'_1 = \{ 7 2 1 1 6 1 1 5 \} \]
\[ \gamma(2) = 9 \quad Q'_2 = 3 \quad S'_2 = \{ 2 2 2 1 0 1 1 5 \} \]
\[ \gamma(3) = 10 \quad Q'_3 = 4 \quad S'_3 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ \gamma(4) = 11 \quad Q'_4 = 5 \quad S'_4 = \{ 2 2 2 1 1 1 1 5 \} \]

Fig. 2. A finite execution sequence of P (and one of P').

\[ a \ b \ c \ d \ u \ v \ w \]
\[ S_0 = \{ 7 2 1 1 6 9 1 5 \} \quad Q_0 = 1 \]
\[ S_1 = \{ 7 2 1 1 6 1 1 5 \} \quad Q_1 = 2 \quad S'_0 = \{ 7 2 1 1 6 1 1 5 \} \]
\[ S_2 = \{ 0 2 1 1 6 1 1 5 \} \quad Q_2 = 3 \quad S'_1 = \{ 0 2 1 1 6 1 1 5 \} \]
\[ S_3 = \{ 1 1 2 1 1 1 1 5 \} \quad Q_3 = 4 \quad S'_2 = \{ 1 1 2 1 1 1 1 5 \} \]
\[ S_4 = \{ 1 1 2 1 1 1 1 5 \} \quad Q_4 = 5 \quad S'_3 = \{ 1 1 2 1 1 1 1 5 \} \]
\[ S_5 = \{ 2 2 2 1 1 1 1 5 \} \quad Q_5 = 6 \quad S'_4 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_6 = \{ 0 2 1 1 0 1 1 5 \} \quad Q_6 = 7 \quad S'_5 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_7 = \{ 2 2 2 1 0 1 1 5 \} \quad Q_7 = 8 \quad S'_6 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_8 = \{ 2 2 2 1 0 1 1 5 \} \quad Q_8 = 9 \quad S'_7 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_9 = \{ 2 2 2 1 1 1 1 5 \} \quad Q_9 = 10 \quad S'_8 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_{10} = \{ 2 2 2 1 1 1 1 5 \} \quad Q_{10} = 11 \quad S'_9 = \{ 2 2 2 1 1 1 1 5 \} \]
\[ S_{11} = \{ 2 2 2 1 1 1 1 23 \} \quad Q_{11} = 12 \quad S'_{10} = \{ 2 2 2 1 1 1 1 5 \} \]

Fig. 3. The finite execution sequence of Σ corresponding to that of P.

within Σ, we must have \( t \leq \theta \leq \theta' \leq \theta' + \lambda' \leq \theta + \lambda \leq t + n \). By Definition 5.3, to say that \( \Sigma' \) is a simple section of \( P \) means that:

- For all \( i, t \leq i \leq \theta' - 1 \) and \( \theta' + \lambda' \leq i \leq t + n - 1 \), if \( c_i(S) = true \), then either \( c_i(S) \leq \theta' \) or \( c_i(S) \geq \theta' + \lambda' \). Therefore, in particular, for all \( i, \theta \leq i \leq \theta' - 1 \) and \( \theta' + \lambda' \leq i \leq \theta + \lambda - 1 \), if \( c_i(S) = true \), then either \( c_i(S) \leq \theta' \) or \( c_i(S) \geq \theta' + \lambda' \).

- For all \( i, \theta' \leq i \leq \theta' + \lambda' - 1 \), if \( c_i(S) = true \), then \( \theta' \leq c_i(S) \leq \theta' + \lambda' \).

Therefore \( \Sigma' \) is a simple section of \( \Sigma \).  □

**Lemma 5.5** Let \( \Sigma \) be a section of \( P \) and let \( \Sigma' \) and \( \Sigma'' \) be simple sections of \( P \), contained within \( \Sigma \) (so that \( \Sigma' \) and \( \Sigma'' \) are simple sections of \( \Sigma \) by Lemma 5.4). Also let \( \Sigma'' \) be adjacent to \( \Sigma' \) within \( P \). Then \( \Sigma'' \) is adjacent to \( \Sigma' \) within \( \Sigma \).
\[ S_0 = \{ 7 0 9 6 9 1 5 \} = Q_0 \quad \gamma(0) = 0 \quad Q_0' = 1 \quad S_0' = \{ 7 0 9 6 9 1 5 \} \\
S_1 = \{ 7 0 9 6 1 1 5 \} = Q_1 \quad \gamma(1) = 1 \quad Q_1' = 2 \quad S_1' = \{ 7 0 9 6 1 1 5 \} \\
S_2 = \{ 0 0 9 6 1 1 5 \} = Q_2 \\
S_3 = \{ 0 0 9 0 1 1 5 \} = Q_3 \\
S_4 = \{ 9 0 9 0 1 1 5 \} = Q_4 \\
S_5 = \{ 9 0 9 -1 1 1 5 \} = Q_5 \\
S_6 = \{ 9 0 9 -1 1 1 5 \} = Q_6 \\
S_7 = \{ 18 0 9 -1 1 1 5 \} = Q_7 \\
S_8 = \{ 18 0 9 -2 1 1 5 \} = Q_8 \\
S_9 = \{ 18 0 9 -2 1 1 5 \} = Q_9 \\
S_{10} = \{ 27 0 9 -2 1 1 5 \} = Q_{10} \\
\text{(abnormal termination; guard is here } S_1(b) > 0, \text{ which is false)} \\
\]

Fig. 4. An infinite execution sequence of \( P \) \( (P' \text{ terminates abnormally).} \)

\[
\begin{array}{|c|c|}
\hline
\text{Step} & \text{Action} \\
\hline
P_1 & u \leftarrow v \\
P_2 & a \leftarrow 0 \\
P_3 & d \leftarrow b \\
P_4 & 4: a \leftarrow a + c \\
P_5 & d \leftarrow d - 1 \\
P_6 & d = 0? \rightarrow 7: \rightarrow 4 \\
P_7 & 7: w \leftarrow u + a \\
P_8 & w \leftarrow w + 1 \\
\hline
\end{array}
\]

Fig. 5. Omitting controllers for sequential statements and unreferenced labels.

**PROOF.** Let \( t, n, \theta, \lambda, \theta', \lambda' \) be as in Lemma 5.4. To say that \( \Sigma'' \) is adjacent to \( \Sigma' \) within \( P \) means that \( \theta' = \theta + \lambda \), by the last sentence of Definition 5.3. But to say that \( \Sigma'' \) is adjacent to \( \Sigma' \) within \( \Sigma \) means exactly the same thing.  \( \square \)

An *induced section* is what happens to an embedded section when a different embedded section is contracted. In order for this to work, the two given sections must be disjoint, in the following sense.

**Definition 5.6** Two sections of a program are *disjoint* if they have no statements in common.

Now suppose that the sequential program \( P \) has two disjoint sections, \( \Sigma \) and \( \Sigma' \), and let \( P' \) be the contraction of \( P \) with respect to \( \Sigma \). Here \( \Sigma' \) appears also to be a section of \( P' \), although certain details differ. Accordingly, we make the following definition.
**Definition 5.7** Using the terminology of Definition 5.5, let $P'$ be the contraction of $P$ with respect to $\Sigma$, and let $\Sigma'$ be another (not necessarily simple) section of $P$, disjoint from $\Sigma$, and having start index $\theta'$ and length $\lambda'$. Then the section $\Sigma''$ of $P'$ which is **induced** by $\Sigma'$ has length $\lambda'$, and is defined as follows. Since $\Sigma$ and $\Sigma'$ are disjoint, there are two cases:

(a) $\theta + \lambda - 1 < \theta'$ (so that $\Sigma$ comes before $\Sigma'$ within $P$). In this case $\Sigma''$ has start index $\theta' + 1 - \lambda$.
(b) $\theta' + \lambda' - 1 < \theta$ (so that $\Sigma'$ comes before $\Sigma$ within $P$). In this case $\Sigma''$ has start index $\theta$. (Note that, in both cases, some controller values might be different, due to the renumbering.)

Simple sections remain simple sections after this operation.

**Theorem 5.4** If $\Sigma''$ is induced by $\Sigma'$, as in Definition 5.7, and $\Sigma'$ is a simple section of $P$, then $\Sigma''$ is a simple section of $P'$.

**Proof.** To say that $\Sigma'$ is a simple section of $P$ means, by Definition 5.3, that if $i$ is such that $c_i^g(S) = true$ and such that:

1. $t \leq i \leq \theta' - 1$, then either $c_i(S) \leq \theta'$ or $c_i(S) \geq \theta' + \lambda'$.
2. $\theta' \leq i \leq \theta' + \lambda' - 1$, then $\theta' \leq c_i(S) \leq \theta' + \lambda'$.
3. $\theta' + \lambda' \leq i \leq t + n - 1$, then either $c_i(S) \leq \theta'$ or $c_i(S) \geq \theta' + \lambda'$.

In Definition 5.7(a), where $\theta + \lambda - 1 < \theta'$ and $\Sigma''$ has start index $\theta' + 1 - \lambda$ and length $\lambda'$, to say that $\Sigma''$ is a simple section of $P'$ means, by Definition 5.3, that if $i$ is such that $c_i^g(S) = true$ and such that the hypothesis of any of the cases (6) through (9) below holds, the corresponding conclusion then holds. (For example, with (7), if $i$ is such that $c_i^g(S) = true$ and $\theta + 1 \leq i \leq \theta' - \lambda$, then either $c_i(S) \leq \theta' + 1 - \lambda$ or $c_i(S) \geq \theta' + \lambda' + 1 - \lambda$.) In Definition 5.7(b), where $\theta' + \lambda' - 1 < \theta$ and $\Sigma''$ has start index $\theta'$ and length $\lambda'$, to say that $\Sigma''$ is a simple section of $P'$ means, by Definition 5.3, that if $i$ is such that $c_i^g(S) = true$ and such that the hypothesis of any of the cases (10) through (13) below holds, the corresponding conclusion then holds.

In all these cases, we need to assume that $i$ is such that $c_i^g(S) = true$. By Definition 5.5(d), setting $j = \kappa(i)$, we have $c_i^g(S) = c_{\kappa(i)}^g(S) = c_j^g(S)$, and thus $c_j^g(S) = c_i^g(S) = \kappa^{-1}(c_{\kappa(i)}^g(S)) = \kappa^{-1}(c_j(S))$. Hence $c_j(S)$ is in the domain of $\kappa^{-1}$; by Lemma 5.2(f), therefore, there are two subcases:

4. $t \leq c_j(S) \leq \theta$. Thus $\kappa^{-1}(c_j(S)) = c_j(S)$ (by Lemma 5.2(h)) = $c_i^g(S)$.
5. $\theta + \lambda \leq c_j(S) \leq t + n$. Thus $\kappa^{-1}(c_j(S)) = c_j(S) + 1 - \lambda$ (again by Lemma 5.2(h)) = $c_i^g(S)$.

42
We now investigate subcases (4) and (5) in each of the main cases (6) through (13), as follows. For Definition 5.7(a):

- If \( t \leq i \leq \theta' + 1 - \lambda - 1 = \theta' - \lambda \), then either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + 1 - \lambda + \lambda = \theta' + \lambda + 1 - \lambda \). We split this into two subcases, as follows:

(6) If \( t \leq i \leq \theta \), then either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + \lambda + 1 - \lambda \).

By Definition 5.4, we have \( j = \kappa(i) = i \). Also, \( t \leq i \leq \theta - 1 \) (since \( \theta + 1 - \lambda - 1 \leq \theta < \theta' \) by Definition 5.7(a)). By (1) above, therefore, either \( c_i(S) \leq \theta' \) or \( c_i(S) \geq \theta' + \lambda ' \). In case (4), if \( c_i(S) \leq \theta' \), then \( c_i'(S) = c_i(S) \leq \theta < \theta' + 1 - \lambda \); if \( c_i(S) \geq \theta' + \lambda ' \), then \( c_i'(S) = c_i(S) \geq \theta' + \lambda ' \geq \theta' + \lambda + 1 - \lambda \). In case (5), if \( c_i(S) \leq \theta' \), then \( c_i'(S) = c_i(S) + 1 - \lambda \leq \theta' + 1 - \lambda \); if \( c_i(S) \geq \theta' + \lambda ' \), then \( c_i'(S) = c_i(S) + 1 - \lambda \geq \theta' + \lambda ' + 1 - \lambda \).

(7) If \( \theta + 1 \leq i \leq \theta' - \lambda \), then either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + \lambda ' + 1 - \lambda \).

By Definition 5.4, we have \( j = \kappa(i) = i + \lambda - 1 \), since \( \theta + 1 \leq i \leq \lambda - 1 \). Since \( \theta + 1 \leq i \leq \theta' - \lambda \), we have \( \lambda + 1 - \lambda = \theta' + 1 - \lambda - 1 \leq i + \lambda - 1 = j \leq \theta' + 1 - \lambda + 1 = \theta' + 1 - \lambda \). In case (4), we have \( c_i'(S) = c_j(S) \leq \theta < \theta' + 1 - \lambda \). In case (5), since \( \theta + 1 \leq \lambda \leq c_j(S) \), we have \( \theta + 1 = \theta + \lambda + 1 - \lambda \leq c_j(S) + 1 - \lambda = c_i'(S) \). Setting \( i = j \) in (1) above, therefore, since \( c_j(S) = \text{true} \) and since \( t \leq \theta \leq \lambda \leq j \leq \theta' - 1 \), we have that either \( c_j(S) \leq \theta' \) (in which case \( c_j(S) = c_j(S) + 1 - \lambda \leq \theta' + 1 - \lambda \)) or \( c_j(S) \geq \theta' + \lambda ' \) (in which case \( c_j(S) = c_j(S) + 1 - \lambda \geq \theta' + \lambda ' + 1 - \lambda \)). Hence either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + \lambda ' + 1 - \lambda \).

(8) If \( \theta' + 1 - \lambda \leq i \leq \theta' + 1 - \lambda + \lambda ' - 1 = \theta' + \lambda + \lambda ' - 1 \), then \( \theta' + 1 - \lambda \leq c_i'(S) \leq \theta' + 1 - \lambda + \lambda ' \). By Definition 5.4, we have \( j = \kappa(i) = i + \lambda - 1 \), since \( \theta < \theta' + 1 - \lambda - i \) and therefore \( \theta + 1 \leq i \)

Since \( \theta + 1 - \lambda \leq i \leq \theta' - \lambda + \lambda ' \), we have \( \theta' \leq i + \lambda - 1 = j \leq \theta' - \lambda + \lambda ' + 1 - \lambda = \theta' + 1 - \lambda \). Therefore we may set \( i = j \) in (2) above, and thus \( \theta' \leq c_j(S) \leq \theta' + \lambda ' \). In case (4), since \( \theta' \leq c_j(S) \), we have \( \theta' + 1 - \lambda \leq \theta' \leq c_j(S)(= c_i'(S)) \leq \theta < \theta' + 1 - \lambda \leq \theta' + 1 - \lambda + \lambda ' \). In case (5), since \( \theta' \leq c_j(S) \leq \theta' + \lambda ' \), we have \( \theta' + 1 - \lambda \leq c_j(S) + 1 - \lambda = c_i'(S) \leq \theta' + 1 - \lambda + \lambda ' \).

(9) If \( \theta' + 1 - \lambda + \lambda ' \leq i \leq t + n - \lambda \), then either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + 1 - \lambda + \lambda ' \). By Definition 5.4, we have \( j = \kappa(i) = i + \lambda - 1 \), since \( \theta \leq \theta' + 1 - \lambda < \theta' + 1 - \lambda + \lambda ' \leq i \), so that \( \theta + 1 \leq i \) and \( \theta' + 1 - \lambda + \lambda ' \leq i \). Since \( \theta' + 1 - \lambda + \lambda ' \leq i \leq t + n - \lambda \), we have \( \theta' + \lambda ' \leq i + \lambda - 1 = j \leq t + n - \lambda + \lambda - 1 = t + n - 1 \). Therefore we may set \( i = j \) in (3) above, and so either \( c_j(S) \leq \theta' \) or \( c_j(S) \geq \theta' + \lambda ' \). In case (4), if \( c_j(S) \leq \theta' \), then \( c_i'(S) = \kappa^{-1}(c_j(S)) = c_j(S) \leq \theta < \theta' + 1 - \lambda \). In case (5), since \( c_j(S) \geq \theta + \lambda \geq \theta + 1 \), we have \( c_i'(S) = \kappa^{-1}(c_j(S)) = c_j(S) + 1 - \lambda \) (by Lemma 5.2(h)) \leq \theta' + 1 - \lambda \) (since \( c_j(S) \leq \theta' \)). In either case (4) or (5), if \( c_j(S) \geq \theta' + \lambda ' \) \( \geq \theta' \geq \theta + \lambda \geq \theta + 1 \), we have \( c_i'(S) = \kappa^{-1}(c_j(S)) = c_j(S) + 1 - \lambda \) (by Lemma 5.2(h)) \geq \theta' + 1 - \lambda + \lambda ' \) (since \( c_j(S) \geq \theta' + \lambda ' \)). Hence either \( c_i'(S) \leq \theta' + 1 - \lambda \) or \( c_i'(S) \geq \theta' + 1 - \lambda + \lambda ' \).
For Definition 5.7(b):

(10) If \( t \leq i \leq \theta' - 1 \), then either \( c'_i(S) \leq \theta' \) or \( c'_i(S) \geq \theta' + \lambda' \). By Definition 5.4, we have \( j = \kappa(i) = i \), since \( t \leq i \leq \theta' - 1 \leq \theta' + \lambda' - 1 \leq \theta \). By (1) above, either \( c_i(S) \leq \theta' \) or \( c_i(S) \geq \theta' + \lambda' \). In case (4), either \( c'_i(S) = c_i(S) \leq \theta' \) or \( c'_i(S) = c_i(S) \geq \theta' + \lambda' \). In case (5), we have \( c'_i(S) = c_i(S) + 1 - \lambda \geq \theta + \lambda + 1 - \lambda = \theta + 1 \geq \theta' + \lambda' \) (by Definition 5.7(b)). Hence either \( c'_i(S) \leq \theta' \) or \( c'_i(S) \geq \theta' + \lambda' \).

(11) If \( \theta' \leq i \leq \theta' + \lambda' - 1 \), then \( \theta' \leq c'_i(S) \leq \theta' + \lambda' \). By Definition 5.4, we have \( j = \kappa(i) = i \), since \( t \leq i \leq \theta' \leq \theta' + \lambda' - 1 \leq \theta \). By (2) above, we have \( \theta' \leq c_i(S) \leq \theta' + \lambda' \). In case (4), we have \( \theta' \leq c'_i(S) = c_i(S) \leq \theta' + \lambda' - 1 \). In case (5), we have \( c'_i(S) = c_i(S) + 1 - \lambda \), and thus \( \theta' + 1 - \lambda \leq c'_i(S) = c_i(S) + 1 - \lambda \leq \theta' + \lambda' \). Also, \( \theta + \lambda \leq c_i(S) \), so that \( \theta' \leq \theta + \lambda \leq \theta + 1 \) (by Definition 5.7(b)) = \( \theta + 1 - \lambda \leq c_i(S) + 1 - \lambda = c'_i(S) \). Hence \( \theta' \leq c'_i(S) \leq \theta' + \lambda' \).

- If \( \theta' + \lambda' \leq i \leq t + n + 1 - \lambda - 1 = t + n - \lambda \), then either \( c'_i(S) \leq \theta' \) or \( c'_i(S) \geq \theta' + \lambda' \). We split this into two subcases, as follows:

(12) If \( \theta' + \lambda' \leq i \leq \theta \), then either \( c'_i(S) \leq \theta' \) or \( c'_i(S) \geq \theta' + \lambda' \). By Definition 5.4, we have \( j = \kappa(i) = i \) because \( t \leq \theta' \leq \theta' + \lambda' \leq i \leq \theta \). By (3) above, either \( c_i(S) \leq \theta' \) or \( c_i(S) \geq \theta' + \lambda' \). In case (4), either \( c'_i(S) = c_i(S) \leq \theta' \) or \( c'_i(S) = c_i(S) \geq \theta' + \lambda' \). In case (5), we have \( c'_i(S) = c_i(S) + 1 - \lambda \). Since \( \theta' + 1 + \lambda - 1 \leq \theta + \lambda - 1 = j \leq t + n - \lambda + \lambda - 1 = t + n - 1 \), therefore, setting \( i = j \) in (3) above, we have that either \( c_j(S) \leq \theta' \) or \( c_j(S) \geq \theta' + \lambda' \). If \( c_j(S) \leq \theta' \) (in either case (4) or case (5)), then, by Definition 5.4, we have \( \kappa(c_j(S)) = c_j(S) \), and hence also \( \kappa^{-1}(c_j(S)) = c_j(S) \). Therefore \( c'_i(S) = \kappa^{-1}(c_j(S)) = c_j(S) \leq \theta' \). Now suppose that \( c_j(S) \geq \theta' + \lambda' \). In case (4), we have \( c'_j(S) = c_j(S) \geq \theta' + \lambda' \). In case (5), since \( c_j(S) \geq \theta + \lambda \geq \theta + 1 \), we have \( \kappa^{-1}(c_j(S)) = c_j(S) + 1 - \lambda \) (by Lemma 5.2(h)) \( \geq \theta + \lambda + 1 - \lambda \geq \theta + 1 \). Therefore \( c'_i(S) = \kappa^{-1}(c_j(S)) \geq \theta + 1 \geq \theta' + \lambda' \). Hence either \( c'_i(S) \leq \theta' \) or \( c'_i(S) \geq \theta' + \lambda' \).

Therefore \( \Sigma'' \) is a simple section of \( P' \). \( \Box \)

The process of taking an induced section, like the operation of contraction, preserves the effect and its guard.

**Theorem 5.5** If the section \( \Sigma'' \) of \( P' \) is induced by the section \( \Sigma' \) of \( P \), as in Theorem 5.4, then \((e_{\Sigma''}, e_{\Sigma''}^2) = (e_{\Sigma'}, e_{\Sigma'}^2)\).
**PROOF.** As in Theorem 5.3, we assume that every statement \( P_i \) in \( P \) has an effect \( e_i \) and a controller \( c_i \), and that every statement \( P'_i \) in \( P' \) has an effect \( e'_{i} \) and a controller \( c'_{i} \). The respective overall effects of \( \Sigma' \) and of \( \Sigma'' \) are denoted here by \( e_{\Sigma'} \) and \( e_{\Sigma''} \) (not \( e_{\Sigma'} \)). By Definition 2.14, we need to show that \( e_{\Sigma'} = e_{\Sigma''} \) and that \( e_{\Sigma'}(S) = e_{\Sigma''}(S) \) whenever \( e_{\Sigma'}(S) = \text{true} \). In Definition 5.7(a), \( \Sigma'' \) has start index \( \theta'' + 1 - \lambda \); in Definition 5.7(b), \( \Sigma'' \) has start index \( \theta' \). In both cases, \( \Sigma'' \) has start index \( \theta'' = \kappa^{-1}(\theta') \), using \( \kappa \) as in Definition 5.4. Now let \( S \in S \). We construct the execution sequences \((S'_0, Q'_0) = (S, \theta'), (S'_1, Q'_1), \ldots \) of \( \Sigma' \) within \( P \), and \((S''_0, Q''_0) = (S, \theta''), (S''_1, Q''_1), \ldots \) of \( \Sigma'' \) within \( P' \). For each \( i \) for which \((S'_i, Q'_i)\) is defined, we shall prove by induction on \( i \) that:

- \((S'_i, Q'_i)\) is also defined;
- \(S''_i = S'_i\); and
- \(Q''_i = \kappa^{-1}(Q'_i)\).

This holds for \( i = 0 \), where \((S'_0, Q'_0)\) is defined, since:

- \((S''_0, Q''_0)\) is also defined;
- \(S''_0 = S = S'_0\); and
- \(Q''_0 = \theta'' = \kappa^{-1}(\theta') = \kappa^{-1}(Q'_0)\).

We assume that this holds for \( i = j \), so that, if \((S'_j, Q'_j)\) is defined, then:

(A) \((S''_j, Q''_j)\) is also defined;
(B) \(S''_j = S'_j\); and
(C) \(Q''_j = \kappa^{-1}(Q'_j)\),

and prove it for \( i = j + 1 \). Therefore we assume that \((S''_{j+1}, Q''_{j+1})\) is defined, so that the conditions of Theorem 5.2(d) hold for \((S'_j, Q'_j)\). Hence \(Q'_j \neq t + n\), so that \( t \leq Q'_j \leq t + n - 1 \); \( e_{k'}(S'_j) = c_{k'}(S'_j) = \text{true} \), where \( k' = Q'_j \); \( S'_{j+1} = e_{k'}(S'_j) \); and \( Q'_{j+1} = e_{k'}(S'_j) \). Setting \( k'' = Q''_j \), we note that \( k'' = Q''_j = \kappa^{-1}(Q'_j) \) (by (C) above) = \( k'(k'') \), so that \( k' = \kappa(k'') \). We can now show that the conditions of Theorem 5.2(d) also hold for \((S''_j, Q''_j)\); that is to say:

- \(Q''_j \neq t + n + 1 - \lambda \). If we had \(Q''_j = t + n + 1 - \lambda \), then \( \kappa(Q''_j) = \kappa(t + n + 1 - \lambda) = t + n \) (by Lemma 5.2(b)). Since \( Q''_j = \kappa^{-1}(Q'_j) \) (by (C) above), we would then have \( Q'_j = \kappa(Q''_j) = t + n \); but we have already shown that \( Q'_j \neq t + n \). Thus \( t \leq Q''_j \leq t + n - \lambda \).
- \( e_{k''}(S''_j) = \text{true} \). Setting \( i = k'' \) in Definition 5.5(c), we have that \((e_{k''}, e_{k''}) = (e_{k''}, e_{k''}) \) (since \( k'' = \kappa(k'') \)). Thus \( e_{k''}(S''_j) = e_{k''}(S'_j) = (S''_j = S'_j) \) by (B) above) = \( e_{k''}(S''_j) = \text{true} \).
- \( c_{k''}(S''_j) = \text{true} \). Setting \( i = k'' \) in Definition 5.5(d), we have that \( c_{k''}(S''_j) = c_{k''}(S''_j) = c_{k''}(S''_j) \) since \( k'' = \kappa(k'') \) = \( c_{k''}(S''_j) \) since \( S''_j = S'_j \) by (B) above) = \( true \).

We can now complete the inductive step above, and show that:
(A') \((S''_{j+1}, Q''_{j+1})\) is also defined. In fact, since the conditions of Theorem 5.2(d) are satisfied, we have \(S''_{j+1} = e_k'(S''_j)\) and \(Q''_{j+1} = c_k'(S''_j)\).
(B') \(S''_{j+1} = S'_{j+1}\). In fact, \(S''_{j+1} = e_k'(S''_j) = e_k'(S'_j)\) (since \(e_k' = e_k\)) (since \(S''_j = S'_j\) by (B) above) = \(S'_{j+1}\).
(C') \(Q''_{j+1} = \kappa^{-1}(Q'_{j+1})\). Setting \(i = k''\) and \(S = S''_j\) in Definition 5.5(e), since \(c_k'(S''_j) = \text{true}\), we have \(c_k'(S''_j) = \kappa^{-1}(c_k(k'')(S''_j))\) (since \(k' = \kappa(k'')\)) (since \(S''_j = S'_j\)) = \(\kappa^{-1}(Q'_{j+1})\). Thus \(Q''_{j+1} = c_k'(S''_j) = \kappa^{-1}(Q'_{j+1})\).

If \(e_{\Sigma'}(S) = \text{false}\), we now show that \(e_{\Sigma''}(S) = \text{false}\) (= \(e_{\Sigma'}(S)\)). This will happen under the following conditions:

- The \((S'_i, Q'_i)\) are defined for all \(i, 0 \leq i < \infty\). In that case the \((S''_i, Q''_i)\) are also defined for all \(i, 0 \leq i < \infty\), so that \(e_{\Sigma'}(S) = e_{\Sigma''}(S) = \text{false}\) by the last part of Definition 5.2.
- \(e_k'(S'_j) = \text{false}\). In this case \(e_k'(S''_j) = e_k'(S'_j) = \text{false}\), so that \(e_{\Sigma'}(S) = e_{\Sigma''}(S) = \text{false}\) by Definition 5.2(b).
- \(c_k'(S'_j) = \text{false}\). In this case \(c_k'(S''_j) = c_k'(S'_j) = \text{false}\), so that \(e_{\Sigma'}(S) = e_{\Sigma''}(S) = \text{false}\) by Definition 5.2(c).

Finally we treat the case in which \(e_{\Sigma'}(S) = \text{true}\). Here the condition of Definition 5.2(a) is satisfied for \(\Sigma'\) and for some \(i\), so that \(Q'_i = t + n\) and \(e_{\Sigma'}(S) = S'_i\). Thus \(t + n = Q'_i = \kappa(Q''_j)\), so that \(Q''_j = \kappa^{-1}(t + n) = t + n + 1 - \lambda\) (by Lemma 5.2(b)). Hence the condition of Definition 5.2(a) is also satisfied for \(\Sigma''\), so that \(e_{\Sigma''}(S) = \text{true}\) (= \(e_{\Sigma'}(S)\)) and \(e_{\Sigma''}(S) = S''_i = S'_i = e_{\Sigma'}(S)\).

5.4 Relocation and Expansion

The inverse of contraction is expansion, or reconstructing a program \(P\) from its contracted program \(P'\). In doing this, we must have what will become the simple section \(\Sigma\) of \(P\) that corresponds to the contracted statement of \(P'\). There is an immediate problem here, in that we might have something like \(\Sigma\), but not with the start index we need to fit it into \(P\). What we need is a section \(\Sigma'\), like \(\Sigma\) except that it has a different start index, and having the same effect (and its guard) as \(\Sigma\). In machine language, this process is known as relocation; here we define relocation in more general terms.

**Definition 5.8** Given a sequential program \(\Sigma\) having the statements \(\Sigma_0, \ldots, \Sigma_{\theta+\lambda-1}\) with respective effects \(e_0, \ldots, e_{\theta+\lambda-1}\) and controllers \(c_0, \ldots, c_{\theta+\lambda-1}\), the **relocation of \(\Sigma\) to \(\theta'\)** is a sequential program \(\Sigma'\) having the statements \(\Sigma'_{\theta'}, \ldots, \Sigma'_{\theta'+\lambda-1}\) with respective effects \(e'_{\theta'}, \ldots, e'_{\theta'+\lambda-1}\) and controllers \(c'_{\theta'}, \ldots, c'_{\theta'+\lambda-1}\), such that for all \(i, 0 \leq i \leq \lambda - 1\), we have:

(a) \((e'_{\theta'+i}, e'_{\theta'+i}) = (e_{\theta+i}, e_{\theta+i})\);
(b) \( c_{\varphi+i}^g = c_{\varphi+i}^g \); and
(c) \( c_{\varphi+i}(S) = c_{\varphi+i}(S) + \theta' - \theta \) whenever \( c_{\varphi+i}(S) = \text{true} \).

Clearly \( \Sigma \) has start index \( \theta \); \( \Sigma' \) has start index \( \theta' \); and both \( \Sigma \) and \( \Sigma' \) have length \( \lambda \). The following lemma will be used in the proof of Theorem 5.7 below.

**Lemma 5.6** In Definition 5.8 above, if \( \theta' = \theta \), then \( \Sigma' = \Sigma \).

This follows immediately from the definition.

**Theorem 5.6** The relocation of \( \Sigma \) to \( \theta' \) has the same effect (and its guard) as \( \Sigma \) does.

**Proof.** We use the terminology of Definition 5.8. Here \( \Sigma \) has the (overall) effect \( e_\Sigma \) with guard \( e_\Sigma^g \), while \( \Sigma' \) has the effect \( e_{\Sigma'} \) with guard \( e_{\Sigma'}^g \). By Definition 2.14, we need to show that \( e_{\Sigma'}^g = e_\Sigma^g \) and that \( e_{\Sigma'}(S) = e_{\Sigma}(S) \) whenever \( e_{\Sigma}(S) = \text{true} \). Now let \( S \in \mathcal{S} \). We construct the execution sequences \((S_0, Q_0) = (S, \theta), (S_1, Q_1), \ldots \) of \( \Sigma \), and \((S_0', Q_0') = (S, \theta'), (S_1', Q_1'), \ldots \) of \( \Sigma' \). For each \( i \) for which \((S_i, Q_i)\) is defined, we shall prove by induction on \( i \) that:

- \((S_i', Q_i')\) is also defined;
- \( S_i' = S_i \); and
- \( Q_i' = Q_i + \theta' - \theta \).

This holds for \( i = 0 \), where \((S_0, Q_0)\) is defined, since:

- \((S_0', Q_0')\) is also defined;
- \( S_0' = S = S_0 \); and
- \( Q_0' = \theta' = \theta + \theta' - \theta = Q_0 + \theta' - \theta \).

We assume that this holds for \( i = j \), so that, if \((S_j, Q_j)\) is defined, then:

- \((S'_j, Q'_j)\) is also defined;
- \( S'_j = S_j \); and
- \( Q'_j = Q_j + \theta' - \theta \).

and prove it for \( i = j + 1 \). Therefore we assume that \((S_{j+1}, Q_{j+1})\) is defined, so that the conditions of Theorem 5.2\((d)\) hold for \((S_j, Q_j)\). Hence \( Q_j \neq \theta + \lambda \), so that \( \theta \leq Q_j \leq \theta + \lambda - 1 \); \( e_k^g(S_j) = e_k^g(S_j) = \text{true} \), where \( k = Q_j \); \( S_{j+1} = e_k(S_j) \); and \( Q_{j+1} = c_k(S_j) \).

We now set \( k' = Q'_j \), and note that \( k' = Q'_j = Q_j + \theta' - \theta \) (by \((C)\) above) = \( k + \theta' - \theta \), so that \( k = k' + \theta - \theta' \). Looking at the conditions of Theorem 5.2\((d)\) for \((S'_j, Q'_j)\), we note that the first of these is \( Q'_j \neq \theta' + \lambda \). If \( k' = Q'_j = \theta' + \lambda \), then we would have \( Q_j = k = k' + \theta - \theta' = \theta' + \lambda + \theta - \theta' = \theta + \lambda \); but we have already shown that \( Q_j \neq \theta + \lambda \). Thus \( \theta' \leq k' = Q'_j \leq \theta' + \lambda - 1 \), so
that $k' = \theta' + i$ where $0 \leq i \leq \lambda - 1$. Therefore we may set $\theta' + i = k'$ in Definition 5.8, so that $\theta + i = \theta' + i + \theta - \theta' = k' + \theta - \theta' = k$. Hence:

- By Definition 5.8(a), we have $(e'_k, e'_k) = (e_k, e_k)$. Thus $e'_k(S'_j) = e'_k(S'_j) = e'_k(S_j)$ (because $S'_j = S_j$, by (B) above) = true; and thus $e'_k(S'_j) = e_k(S_j)$.
- By Definition 5.8(b), we have $c'_k(S'_j) = c'_k$. Thus $e'_k(S'_j) = e'_k(S'_j) = e'_k(S_j)$ (because $S'_j = S_j$ = true).
- By Definition 5.8(c), we have $c'_k(S'_j) = c_k(S'_j) + \theta' - \theta$, since $c'_k(S'_j) = true$.

Thus the conditions of Theorem 5.2(d) also hold for $(S'_j, Q'_j)$. We can now complete the inductive step above, and show that:

(A') $(S'_{j+1}, Q'_{j+1})$ is also defined. In fact, since the conditions of Theorem 5.2(d) are satisfied, we have $S'_{j+1} = e'_k(S'_j)$ and $Q'_{j+1} = c'_k(S'_j)$.
(B') $S'_{j+1} = S_{j+1}$. In fact, $S'_{j+1} = e'_k(S'_j) = e_k(S_j) = S_{j+1}$.
(C') $Q'_{j+1} = Q_{j+1} + \theta' - \theta$. In fact, $Q'_{j+1} = c'_k(S'_j) = c_k(S_j) + \theta' - \theta$ (by (C) above) = $Q_{j+1} + \theta' - \theta$.

If $e^3_\Sigma(S) = false$, we now show that $e^3_\Sigma(S) = false$ ($= e^3_\Sigma(S)$). This will happen under the following conditions:

- The $(S_i, Q_i)$ are defined for all $i, 0 \leq i < \infty$. In that case the $(S'_i, Q'_i)$ are also defined for all $i, 0 \leq i < \infty$, so that $e^3_\Sigma(S) = e^3_\Sigma(S) = false$ by the last part of Definition 5.2.
- $e^3_k(S_j) = false$. In this case $e^3_k(S'_j) = e^3_k(S_j) = false$, so that $e^3_\Sigma(S) = e^3_\Sigma(S) = false$ by Definition 5.2(b).
- $c^3_k(S_j) = false$. In this case $c^3_k(S'_j) = c^3_k(S_j) = false$, so that $e^3_\Sigma(S) = e^3_\Sigma(S) = false$ by Definition 5.2(c).

Finally we treat the case in which $e^3_\Sigma(S) = true$. Here the condition of Definition 5.2(a) is satisfied for $\Sigma$ and for some $i$, so that $Q_i = \theta + \lambda$ and $e_\Sigma(S) = S_i$. Thus $\theta + \lambda = Q_j = Q'_j + \theta - \theta'$, so that $Q'_j = Q_j + \theta' - \theta = \theta + \lambda + \theta' - \theta = \theta' + \lambda$. Hence the condition of Definition 5.2(a) is also satisfied for $\Sigma'$, so that $e^3_\Sigma'(S) = true$ ($= e^3_\Sigma(S)$) and $e_\Sigma'(S) = S'_i = S_i = e_\Sigma(S)$. □

We can now define an expansion of a sequential program, given a simple section of the expansion which might have to be relocated.

**Definition 5.9** Given:

- a sequential program $P'$ having the statements $P'_1, \ldots, P'_{t+n'–1}$ with respective effects $e'_1, \ldots, e'_{t+n'–1}$ and controllers $c'_1, \ldots, c'_{t+n'–1}$, together with
- a sequential statement $P'_0$ of $P'$ (that is, $(c'_0(S), c'_0(S)) \equiv (\theta + 1, true)$ by Definition 5.1(b)) with effect $e_\theta$, and

48
• a sequential program $\Sigma$, having start index $\theta'$ and length $\lambda$, such that $(e_\Sigma, e_\Sigma^\theta) = (e_\theta, e_\theta^\theta)$,

the expansion of $P'$ by $\Sigma$ at $\theta$ is a sequential program $P''$ with statements $P''_1, \ldots, P''_{n' + \lambda - 1}$ (where $n = n' + \lambda - 1$) with respective effects $e_i', \ldots, e_{i+n-1}'$ and controllers $c_i', \ldots, c_{i+n-1}'$ defined as follows. Let $\Sigma'$ be the relocation of $\Sigma$ to $\theta$, as in Definition 5.8, so that $e_{\Sigma'} = e_\Sigma$ by Theorem 5.6. Furthermore let $\Sigma'$ have statements $\Sigma_\theta, \ldots, \Sigma_{\theta+\lambda-1}$, with respective effects $e_\theta', \ldots, e_{\theta+\lambda-1}'$ and controllers $c_\theta', \ldots, c_{\theta+\lambda-1}'$. Then:

(a) For all $i, t \leq i \leq \theta - 1$, we have $(e_i', e_i'^g) = (e_i, e_i^g)$ and $(c_i', c_i'^g) = (c_i, c_i^g)$.

(b) For all $i, \theta \leq i \leq \theta + \lambda - 1$, we have $(e_i', e_i'^g) = (e_i^*, e_i^g)$ and $(c_i', c_i'^g) = (c_i^*, c_i^g)$. (All effects and controllers within $\Sigma$ have already been relocated.)

(c) For all $i, \theta + \lambda \leq i \leq t + n - 1$, we have $(e_i', e_i'^g) = (e_{i+1-\lambda}', e_{i+1-\lambda}^g)$, $c_i'^g = c_{i+1-\lambda}'$, and $c_i'^g(S) = c_{i+1-\lambda}'(S) + \lambda - 1$ if $c_i'(S) = true$. (After $\Sigma$, all effects and controllers are relocated forward.)

Expansion and contraction are inverse operations; we need to prove this in both directions.

**Theorem 5.7** Let $P'$ be the contraction of $P$ with contracted section $\Sigma$, where $\Sigma$ has start index $\theta$ in $P$. Let $P''$ be the expansion of $P'$ by $\Sigma$ at $\theta$. Then $P'' = P$.

**Proof.** We assume that $P$ has length $n$, so that $P'$, by Definition 5.5, has length $n + 1 - \lambda$, and $P''$, by Definition 5.9, has length $n$. We show that $(e_i', e_i'^g) = (e_i, e_i^g)$ and $(c_i', c_i'^g) = (c_i, c_i^g)$ for all $i$ over each of the following ranges:

- $t \leq i \leq \theta - 1$. By Definition 5.4, we have $\kappa(i) = i$ for all $i$ in this range (and thus also $\kappa^{-1}(i) = i$). Therefore $(e_{i}', e_{i}^{g}) = (e_{i}, e_{i}^{g})$ (by Definition 5.5(e)) = $(e_{i}, e_{i}^{g})$. Also, $c_{i}^{g} = c_{\kappa(i)}^{g}$ (by Definition 5.5(d)) = $c_{i}^{g}$; while, whenever $c_{i}^{g}(S) = true$, we have $c_{i}'(S) = \kappa^{-1}(c_{\kappa(i)}(S))$ (by Definition 5.5(e)) = $c_{i}(S)$. Therefore $(e_{i}', c_{i}^{g}) = (e_{i}, c_{i}^{g})$ by Definition 2.14. By Definition 5.9(a), we have $(e_{i}', e_{i}^{g}) = (e_{i}', c_{i}^{g})$ and $(e_{i}', c_{i}^{g}) = (e_{i}', c_{i}^{g})$. Therefore $(e_{i}', c_{i}^{g}) = (e_{i}, c_{i}^{g})$ and $(e_{i}', c_{i}^{g}) = (e_{i}, c_{i}^{g})$.

- $\theta \leq i \leq \theta + \lambda - 1$. This is the range of $\Sigma$ within $P$, so that each $(e_i, e_i^g)$ and $(c_i, c_i^g)$ in this range is within $\Sigma$. This particular $\Sigma$ has start index $\theta'$, but, by hypothesis, $\Sigma$ has start index $\theta'$, so that, here, $\theta' = \theta$. Thus $\Sigma' = \Sigma$ by Lemma 5.6, so that $(e_{i}', e_{i}^{g}) = (e_{i}, e_{i}^{g})$ and $(c_{i}', c_{i}^{g}) = (c_{i}, c_{i}^{g})$. By Definition 5.9(b), we have $(e_{i}', e_{i}^{g}) = (e_{i}', e_{i}^{g})$ and $(e_{i}', c_{i}^{g}) = (c_{i}', c_{i}^{g})$. Therefore $(e_{i}', c_{i}^{g}) = (e_{i}, c_{i}^{g})$ and $(e_{i}', c_{i}^{g}) = (c_{i}, c_{i}^{g})$.

- $\theta + \lambda \leq i \leq t + n - 1$. By Definition 5.4, we have $\kappa(i) = i + \lambda - 1$ for all $i$ in
this range (and thus also $\kappa^{-1}(i) = i + 1 - \lambda$). Therefore $(e'_i, e''_{i}) = (e_{i(i)}, e''_{i(i)})$
(by Definition 5.5(c)) = $(e_{i+1-\lambda}, e''_{i+1-\lambda})$, so that $(e'_i, e''_{i}) = (e_{i}, e''_{i})$.
By Definition 5.9(c), we have $(e''_{i}, e''_{i+1}) = (e'_i, e''_{i+1-\lambda})$; therefore, $(e'_i, e''_{i}) = (e_{i}, e''_{i})$
Also, $c'_{i} = e''_{i+1-\lambda}$ (by Definition 5.5(d)) = $(e_{i+1-\lambda}, e''_{i+1-\lambda})$, so that $c''_{i} = c''_{i+1-\lambda}$ = $c''_{i}$; while, whenever $c''_{i}(S) = \text{true}$, we have $c'_i(S) = \kappa^{-1}(c''_{i}(S))$ (by Definition 5.5(e)) = $c_{i+1-\lambda}(S) + \lambda - 1$. Replacing $i$ by $i + 1 - \lambda$ in the above (so that $i + 1 - \lambda$ is replaced by $i$), we obtain $c'_{i+1-\lambda}(S) = c_i(S) + \lambda - 1$, or $c'_{i+1-\lambda}(S) + 1 - \lambda = c_i(S)$. By Definition 5.9(c), we have $c''_{i} = c''_{i+1-\lambda}$, and $c'_i(S) = c''_{i+1-\lambda}(S) + 1 - \lambda$ if $c''_{i}(S) = \text{true}$. Thus $c''_{i} = c''_{i+1-\lambda} = c''_{i}$; while, whenever $c''_{i}(S) = \text{true}$, we have $c''_{i}(S) = c''_{i+1-\lambda}(S) + 1 - \lambda = c_i(S)$. Therefore $(c''_{i}, c''_{i+1}) = (c_{i}, c_{i+1})$
by Definition 2.14.

Hence $P'' = P$. $\square$

**Theorem 5.8** Let $P'$ be the expansion of $P$ by $\Sigma$ at $\theta$. Let $P''$ be the contraction of $P'$ with contracted section $\Sigma$, where $\Sigma$ has start index $\theta$ in $P$. Then $P'' = P$.

**Proof.** We assume that $P$ has length $n+1-\lambda$, so that $P'$, by Definition 5.9, has length $n$, and $P''$, by Definition 5.5, has length $n + 1 - \lambda$. We show that $(e''_{i}, e''_{i}) = (e_{i}, e''_{i})$ and that $(e''_{i}, e''_{i}) = (c_{i}, c''_{i})$ for all $i$ over each of the following ranges:

- $t \leq i \leq \theta - 1$. By Definition 5.9(a), we have $(e'_i, e''_{i}) = (e_{i}, e''_{i})$ and $(e'_i, e''_{i}) = (c_{i}, c''_{i})$. For the contraction, by Definition 5.4, we have $\kappa(i) = i$ for all $i$ in this range (and thus also $\kappa^{-1}(i) = i$). Therefore $(e''_{i}, e''_{i}) = (e_{i}, e''_{i})$
(by Definition 5.5(c)) = $(e_{i}, e''_{i})$. Also, $c''_{i} = e''_{i}$ (by Definition 5.5(d)) = $c''_{i}$; while, whenever $c''_{i}(S) = \text{true}$, we have $c''_{i}(S) = \kappa^{-1}(c''_{i}(S))$ (by Definition 5.5(e)) = $c''_{i}(S)$. Hence $(c''_{i}, c''_{i+1}) = (c_{i}, c''_{i})$ by Definition 2.14. Therefore $(e''_{i}, e''_{i}) = (e_{i}, e''_{i})$ and $(e''_{i}, e''_{i}) = (c_{i}, c''_{i})$.

- $i = \theta$. By the hypotheses of Definition 5.9, $P_{\theta}$ is a sequential statement of $P$,
with $(e_{i}, e''_{i}) = (e_{\theta}, e''_{\theta})$. By Definition 5.5, $P''_{\theta}$ is the contracted statement of $P''$, which, by Definition 5.5(b), is a sequential statement. Thus $(e''_{\theta}, e''''_{\theta}) = (e_{\theta}, e''_{\theta})$ (by Definition 5.5(a)) = $(e_{\theta}, e''_{\theta})$. Since both $P_{\theta}$ and $P''_{\theta}$ are sequential statements, we have $(e''_{\theta}, e''''_{\theta}) \equiv (\theta + 1, \text{true}) \equiv (e_{\theta}, e''_{\theta})$.

- $\theta + 1 \leq i \leq t + n - \lambda$. By Definition 5.9(c), we have $(e'_i, e''_{i}) = (e_{i+1-\lambda}, e''_{i+1-\lambda})$, so that $(e''_{i+1-\lambda}, e''_{i+1-\lambda}) = (e_{i}, e''_{i})$. For the contraction, by Definition 5.4, we have $\kappa(i) = i + \lambda - 1$ for all $i$ in this range (and thus also $\kappa^{-1}(i) = i + 1 - \lambda$). Hence $(e''_{i}, e''_{i}) = (e''_{i+1-\lambda}, e''_{i+1-\lambda})$ (by Definition 5.5(c)) = $(e_{i+1-\lambda}, e''_{i+1-\lambda})$. Therefore, $(e''_{i}, e''_{i}) = (e_{i}, e''_{i})$
Also, by Definition 5.9(c), we have $c''_{i} = c''_{i+1-\lambda}$, and $c''_{i}(S) = c''_{i+1-\lambda}(S) + 1 - \lambda$ if $c''_{i}(S) = \text{true}$. Replacing $i$ by $i + 1 - \lambda$ in the above (so that $i + 1 - \lambda$
is replaced by \( i \), we obtain \( c_{i+1}^q = c_i^q \), and \( c_{i+\lambda-1}'(S) = c_i(S) + 1 - \lambda \) (so that \( c_{i+\lambda-1}'(S) + \lambda - 1 = c_i(S) \)) if \( c_i^q(S) = \text{true} \). For the contraction, we have \( c_i'^q = c_{\kappa(i)}^q \) (by Definition 5.5(d)) = \( c_{i+\lambda-1}' \), so that \( c_i'^{\lambda}(S) = c_i^{\lambda} \); while, whenever \( c_i'^{\lambda}(S) = \text{true} \), we have \( c_i'(S) = \kappa^{-1}(c_{\kappa(i)}(S)) \) (by Definition 5.5(e)) = \( c_{i+\lambda-1}'(S) + \lambda - 1 \). Thus \( c_i'^{\lambda} = c_{i+\lambda-1}'^q \); while, whenever \( c_i'^{\lambda}(S) = \text{true} \), we have \( c_i'(S) = c_{i+\lambda-1}'(S) + \lambda - 1 = c_i(S) \). Therefore \((c_i', c_i'^{\lambda}) = (c_i, c_i^q)\) by Definition 2.14.

Hence \( P'' = P. \) \( \square \)

Theorem 5.3 shows that contraction of a sequential program preserves its effect and its guard. We may now combine this with Theorem 5.8 to show that the same is true of expansion.

**Theorem 5.9** If \( P'' \) is the expansion of \( P' \) by \( \Sigma \) at \( \theta \), as in Definition 5.9, then \((e_{P''}, e_{P''}^q) = (e_P, e_P^q)\).

**PROOF.** Let \( P \) be the contraction of \( P'' \) with contracted section \( \Sigma \). By Theorem 5.8, \( P = P' \), so that \((e_P, e_P^q) = (e_P, e_P^q)\). By Theorem 5.3, we have \((e_P, e_P^q) = (e_{P''}, e_{P''}^q)\). Therefore \((e_{P''}, e_{P''}^q) = (e_{P''}, e_{P''}^q)\). \( \square \)

Similarly, we can define sections induced by an expansion, as well as by a contraction. Given an expansion \( P'' \) of \( P' \) by \( \Sigma \) at \( \theta \), by Definition 5.9, and given a second section \( \Sigma' \) of \( P' \), disjoint from \( \Sigma \), the statements of \( P'' \) that correspond, under the expansion, to the statements of \( \Sigma' \) within \( P' \) constitute the section \( \Sigma'' \) of \( P'' \) that is induced by \( \Sigma' \). Where these are, within \( P'' \), will depend on whether \( \Sigma' \) comes before or after \( \Sigma \) within \( P' \). As with sections induced by a contraction, we may show that simple sections remain simple under this operation, and that the process of inducing, described here, preserves the effect of \( \Sigma' \) and its guard. The details are left to the reader.

6 Simple Section Interchange

Suppose now that we have a program \( P \) containing two adjacent simple sections, \( \Sigma \) and \( \Sigma' \), having respective effects \( e_\Sigma \) and \( e_{\Sigma'} \), and we wish to know whether these can be interchanged; that is, whether \( \Sigma' \) can be done first. In order for this to work, \( \Sigma \) followed by \( \Sigma' \) must itself be a simple section. If this were not the case, then a statement somewhere else in \( P \) could go to \( \Sigma' \). After the interchange, \( \Sigma \) would now be done following \( \Sigma' \); whereas the original intention was presumably that \( \Sigma \) not be done at all.
Note that we don’t necessarily know what $e_\Sigma$ and $e_{\Sigma'}$ do, or even whether they are computable, as discussed in Section 5 above. Nevertheless, we can use Theorems 12 and 13 of [9], and Theorems 4.3, 5.1, and 5.2 of this paper, to put bounds on the conditional input and output regions of $e_\Sigma$ and $e_{\Sigma'}$. We can now give conditions under which $e_\Sigma$ and $e_{\Sigma'}$ commute, so that $\Sigma$ and $\Sigma'$ can be interchanged.

The contraction and expansion processes provide an elegant description of the result of interchanging $\Sigma$ and $\Sigma'$. If we tried to describe this result without using contraction or expansion, we would have to express the fact that the statements of $\Sigma$ have been relocated forward, and those of $\Sigma'$ have been relocated backward; and therefore each controller value, which was equal to some position within $\Sigma$, or within $\Sigma'$, is now equal to that position plus the amount of the move. This would be rather unwieldy; and so we now give a more usable definition of this concept. First we need the idea of a simple pair of sequential statements, and of its transpose.

**Definition 6.1** A simple pair is a sequential program $P$ of length 2, and containing two sequential statements $P_t$ and $P_{t+1}$, for some start index $t$, having respective effects $e_t$ and $e_{t+1}$. The transpose of $P$ is the sequential program $P'$ of length 2, and containing two sequential statements $P'_t$ and $P'_{t+1}$, for the same start index $t$, having respective effects $e'_t = e_{t+1}$ and $e'_{t+1} = e_t$. (Note that the controllers here are all determined by Definition 5.1(b).)

The effect of a simple pair is a simple composition.

**Lemma 6.1** If $P$ is a simple pair as in Definition 6.1, then $(e_P, e_P^0) = (e_t, e_t^0) \circ (e_{t+1}, e_{t+1}^0)$.

**Proof.** Let $(e_t, e_t^0) = (e_t, e_t^0) \circ (e_{t+1}, e_{t+1}^0)$. By Definition 2.14, we need to show that $e_P^0 = e_P$, and that $e'_t = e_P(S)$ whenever $e_P^0(S) = true$. By Definition 2.11, we have that $e_P^0(S) = e_t^0(S) \& e_{t+1}^0(e_t(S))$ and that $e'(S) = e_2(e_t(S))$. Now let $S \in S$ and consider the execution sequence $(S_0, Q_0) = (S, t), (S_1, Q_1), \ldots$. Setting $i = 0$ in Definition 5.2, we have $k = Q_i = Q_0 = t \neq t + 2 = t + n$, so Definition 5.2(a) cannot hold. Since $P_t$ is a sequential statement, we have $e_t^0(S) = true$ by Definition 5.1(b), so Definition 5.2(c) cannot hold either. If Definition 5.2(b) holds, then $e_t^0(S_0) = false$ and $e_P^0(S) = false$. We will now assume that Definition 5.2(d) holds, so that $e_t^0(S_0) = true; S_1 = e_t(S_0) = e_t(S);$, and $Q_1 = c_t(S_0) = t + 1$ by Definition 5.1(b). Thus $(S_1, Q_1) = (e_t(S), t + 1)$.

We now set $i = 1$ in Definition 5.2, so that $k = Q_i = Q_1 = t + 1 \neq t + 2 = t + n$; again Definition 5.2(a) cannot hold. Since $P_{t+1}$ is a sequential statement, we have $e_{t+1}(S_1) = true$ by Definition 5.1(b), and again Definition 5.2(c) cannot hold. If Definition 5.2(b) holds, then $e_{t+1}^0(S_1) = false$ and $e_P^0(S) = false$. If
Definition 5.2(d) holds, then $e_{t+1}^Q(S_1) = \text{true}$; $S_2 = e_{t+1}(S_1) = e_{t+1}(e_t(S))$; and $Q_2 = e_{t+1}(S_0) = t + 2$ by Definition 5.1(b). Thus $(S_2, Q_2) = e_{t+1}(e_t(S)), t + 2)$. Setting $i = 2$ in Definition 5.2, we obtain $k = Q_2 = t + 2 = t + n$, so that Definition 5.2(a) now holds; $e_p^Q(S) = \text{true}$; and $e_p^Q(S) = S_2 = e_{t+1}(e_t(S))$.

In the above, we have assumed that $e^Q_i(S_0) = e^Q_i(S) = \text{true}$ and that $e^Q_{t+1}(S_1) = e^Q_{t+1}(e_t(S)) = \text{true}$; that is, that $e^g_i(S) = \text{true}$, and we have shown that $e^g_p(S) = \text{true}$ and $e^g_p(S) = e_{t+1}(e_t(S))$. If $e^g_p(S) = e^g_i(S) = \text{false}$, there are two cases. Either $e^g_p(S_0) = \text{false}$, in which case $e^g_p(S) = \text{false}$, or else $e^g_i(S_0) = \text{true}$ and $e^g_{t+1}(S_1) = \text{false}$, in which case also $e^g_p(S) = \text{false}$. Therefore $e^g_i = e^g_p$, since these are either both true or both false. □

Conditions for the effect of a simple pair being equal to that of its transpose are given by Corollary 2.1.

**Lemma 6.2** Using the terminology of Lemma 6.1, if $\text{COR}(e_t, e^g_i) \cap (\text{CIR}(e_{t+1}, e^g_{t+1}) \cup \text{COR}(e_{t+1}, e^g_{t+1})) = \emptyset$ and $\text{COR}(e_t, e^g_i) \cap (\text{CIR}(e_t, e^g_i) \cup \text{COR}(e_t, e^g_i)) = \emptyset$, then $(e_p, e^g_p) = (e'_p, e^g_p)$.

**PROOF.** The hypotheses above show us that the conditions of Corollary 2.1 are satisfied for $P$, and also, by symmetry, for $P'$. Therefore $(e_p, e^g_p) = (e_t, e^g_i) \circ (e_{t+1}, e^g_{t+1})$ (by Lemma 6.1) = $(e_t, e^g_i) \circ (e_t, e^g_i)$ (by Corollary 2.1) = $(e'_p, e^g_p)$ (again by Lemma 6.1). □

We are now ready to present our main result, as described in Section 1 above.

**Theorem 6.1** If $\Sigma$ and $\Sigma'$ are simple sections of $P$; $U$, which is $\Sigma$ followed by $\Sigma'$, is also a simple section of $P$; $\text{COR}(e_{\Sigma}, e^g_{\Sigma}) \cap (\text{CIR}(e_{\Sigma'}, e^g_{\Sigma'}) \cup \text{COR}(e_{\Sigma'}, e^g_{\Sigma'})) = \emptyset$; and $\text{COR}(e_{\Sigma'}, e^g_{\Sigma'}) \cap (\text{CIR}(e_{\Sigma'}, e^g_{\Sigma'}) \cup \text{COR}(e_{\Sigma'}, e^g_{\Sigma'})) = \emptyset$, then $(e_{P^m}, e^g_{P^m}) = (e_p, e^g_p)$, where $P^m$ is formed from $P$ by interchanging $\Sigma$ and $\Sigma'$ as described below.

**PROOF.** Let $\Sigma$ have start index $\theta$, and let $\Sigma'$ have start index $\theta'$. Here $U$ also has start index $\theta$, since $U$ is $\Sigma$ followed by $\Sigma'$. The process of forming $P^m$ from $P$ is as follows:

1. By Lemma 5.4, $\Sigma$ and $\Sigma'$, which are simple sections of $P$, are also simple sections of $U$. By Lemma 5.5, since $\Sigma'$ is adjacent to $\Sigma$ within $P$, it is also adjacent to $\Sigma$ within $U$. (We are going to form $U'$, which will be $U$ with $\Sigma$ and $\Sigma'$ interchanged.)
2. Form the contraction $V$ of $U$ with contracted section $\Sigma$. By Theorem 5.3, we have $(e_V, e^g_V) = (e_U, e^g_U)$. Since $\Sigma$ and $\Sigma'$ are disjoint, this process
produces an induced section $\Sigma'_1$ of $\Sigma'$, by Definition 5.7. Here $\Sigma'_1$ is a simple section of $V$, by Theorem 5.4; and the effect of $\Sigma'_1$ is the same as that of $\Sigma'$, by Theorem 5.5. By Definition 5.5, the contracted statement of $\Sigma$ is a sequential statement with index $\theta$ and effect $(e_\Sigma, e_\Sigma^g)$.

(3) Form the contraction $W$ of $V$ with contracted (simple) section $\Sigma'_1$. By Theorem 5.3, we have $(e_W, e_W^g) = (e_V, e_V^g)$. By Definition 5.5, the statement of $V$ with index $\theta$ is also a sequential statement of $W$, while the contracted statement of $\Sigma'_1$ has index $\theta + 1$ and effect $(e_{\Sigma'}, e_{\Sigma'}^g)$. Hence $W$ is a simple pair, by Definition 6.1.

(4) Let $W'$ be the transpose of $W$. The conditions of Lemma 6.2 are now satisfied, and thus the effect of $W'$, and its guard, are the same as those of $W$. By Definition 6.1, the effects of the two sequential statements of $W'$ are thus $(e_{\Sigma'}, e_{\Sigma'}^g)$ (at index $\theta$) and $(e_\Sigma, e_\Sigma^g)$ (at index $\theta + 1$).

(5) Expand $W'$ by $\Sigma$ at $\theta + 1$, producing $V'$. By Theorem 5.9, we have $(e_{V'}, e_{V'}^g) = (e_W, e_W^g)$. The sequential statement at index $\theta + 1$, with effect $(e_{\Sigma}, e_{\Sigma}^g)$, has been expanded into a relocation of $\Sigma$.

(6) Expand $V'$ by $\Sigma'$ at $\theta$, producing $U'$. By Theorem 5.9, we have $(e_{U'}, e_{U'}^g) = (e_{V'}, e_{V'}^g)$. Thus $(e_{U'}, e_{U'}^g) = (e_W, e_W^g) = (e_V, e_V^g) = (e_V, e_V^g) = (e_{U'}, e_{U'}^g)$. The sequential statement at index $\theta$, with effect $(e_{\Sigma'}, e_{\Sigma'}^g)$, has been expanded into a relocation of $\Sigma'$, so that $\Sigma$ and $\Sigma'$ have indeed been interchanged.

(7) Form the contraction $P'$ of $P$ with contracted section $U$. By Theorem 5.3, we will have $(e_{P'}, e_{P'}^g) = (e_P, e_P^g)$.

(8) If $W$ starts at $P_\theta$ within $P$, then, by Definition 5.5(b), $P'_\theta$ is a sequential statement of $P'$, with $(e_{\theta'}, e_{\theta'}^g) = (e_{\theta'}, e_{\theta'}^g)$. Thus the conditions of Definition 5.9 are satisfied, and we may form the expansion $P''$ of $P'$ by $U'$ at $\theta$. Thus $(e_{P''}, e_{P''}^g) = (e_{P'}, e_{P'}^g)$ (by Theorem 5.9) = $(e_P, e_P^g)$. □

7 Applying the Main Result

The hypotheses of Theorem 6.1 are more easily usable if they are expressed in the equivalent form

(a) $\text{COR}(e_\Sigma, e_\Sigma^g) \cap \text{CIR}(e_{\Sigma'}, e_{\Sigma'}^g) = \emptyset$ and
(b) $\text{COR}(e_\Sigma, e_\Sigma^g) \cap \text{COR}(e_{\Sigma'}, e_{\Sigma'}^g) = \emptyset$ and
(c) $\text{CIR}(e_\Sigma, e_\Sigma^g) \cap \text{COR}(e_{\Sigma'}, e_{\Sigma'}^g) = \emptyset$.

Applying this theorem involves repeated use of the following Inclusion Principle: If we know that $A \subseteq C$ and $B \subseteq D$, and we need to prove that $A \cap B = \emptyset$, it is sufficient to prove that $C \cap D = \emptyset$. Here (a), (b), and (c) above are each of the form $A \cap B = \emptyset$, and Theorems 2.2, 2.3, 2.4, 2.6, 4.1, 4.2, 4.3, 5.1, and 5.2 are each of the form $A \subseteq C$ or $B \subseteq D$ (or $A = C$ or $B = D$, which respectively imply these). Using these theorems, we can bound each of the
appropriate regions, and then apply our Inclusion Principle.

The following general class of examples should make this clear, in simple cases. Let $P, P', \text{ and } U$ be as in Theorem 6.1. Let $X$ (respectively, $X'$) be the set of all variables which occur in conditional input or output regions of the effects, or in conditional relevant regions of the controllers, of statements of $P$ (respectively, $P'$). Let $Y$ (respectively, $Y'$) be the set of all variables which occur in conditional output regions of statements of $P$ (respectively, $P'$). Then $P$ and $P'$ may be interchanged if there is no variable in both $X$ and $Y'$, or in both $X'$ and $Y$, or in both $Y$ and $Y'$.

In general, we often do not know what $X, X', Y, \text{ and } Y'$ are. However, now let $X_0$ (respectively, $X'_0$) be the set of all variables which can possibly occur in conditional input or output regions (see Theorem 4.3) of the effects, or in conditional relevant regions (see Theorems 2.2, 2.3, 2.4, 2.6, 4.1, and 4.2) of the controllers, of statements of $P$ (respectively, $P'$). Let $Y_0$ (respectively, $Y'_0$) be the set of all variables which can possibly occur in conditional output regions (see Theorem 4.3(a)) of the effects of statements of $P$ (respectively, $P'$). Then, by our Inclusion Principle, $P$ and $P'$ may be interchanged if there is no variable in both $X_0$ and $Y'_0$, or in both $X'_0$ and $Y_0$, or in both $Y_0$ and $Y'_0$.

We may, indeed, exclude from $X, X', X_0, \text{ and } X'_0$ any variables which occur only in conditional output regions. This is because any such variable, if it is in both $X$ and $Y'$, or in both $X'$ and $Y$, must also be in both $Y$ and $Y'$; similarly, if it is in both $X_0$ and $Y'_0$, or in both $X'_0$ and $Y_0$, it must also be in both $Y_0$ and $Y'_0$.

Note that if all variables in $X_0$ and $X'_0$ result from ordinary fetches, and if all variables in $Y_0$ and $Y'_0$ result from ordinary stores (and there is no aliasing), then Theorem 6.1 can still be used, but it is unnecessary because this case is covered by conventional compiler dependence analysis. The real power of the theorem arises from the fact that it may be used in more general situations. In particular, the effects of statements may contain calls to procedures, which do their own fetching, storing, and further procedure calling. However, the sizes of the conditional regions of these procedure calls may be reduced in a number of ways, making it more likely that the theorem can be used. For example:

- A register, or other variable, $x$ that is saved and restored by a program $P$ does not appear in the conditional output region of the effect of $P$, even if it appears in the conditional output regions of individual statements of $P$. This is because the value of $x$ when $P$ ends is always the same as its value when $P$ starts (see Theorems 2.10 and 5.2).
- A register, or temporary variable, $x$ that is properly initialized at the start of a program $P$ does not appear in the conditional input region of the effect of $P$, even if it appears in the conditional input regions of individual
statements of $P$. This is because the value of $x$, before initialization, cannot affect the result of executing $P$, since such a value is overwritten by the initialization (see Theorems 2.12 and 5.2).

- A register, or variable, $x$ might be in the conditional input or output region of the effect $e$ of $P$ only if some condition $g$ does not hold. Including $g$ within the guard of $e$ would effectively exclude $x$ from that region.
- Finally, a proof of correctness of $P$ can include the proof that some variable is not in the conditional input, or the conditional output, region of its effect. As an example, if $x = x_0$ is part of the entry assertion of $P$ (where $x_0$ is an artificial variable), and $x = x_0$ is also part of the exit assertion of $P$, then $x$ is not in the conditional output region of $P$, where the applicable guard is the entry assertion of $P$.

One advantage of interchanging two adjacent simple sections arises when there are three such sections, $P$, $P'$, and $P''$, in sequence. If $P$ and $P'$ are interchanged, or if $P'$ and $P''$ are interchanged, $P''$ will then immediately follow $P$, and this might imply that some redundancy in $P''$ can be eliminated. Such redundancy would normally not be noticed without the interchange.

8 Related Work and Future Work

The conditions presented here, as to when two adjacent simple sections can be interchanged, are the extensions, to conditional regions, of the conditions on ordinary input and output regions described in Corollary 5.4 of [7], published in 1966. A restricted form of this theorem was discovered independently, four months later, by Bernstein [3], and the hypotheses of the theorem, recast in modern form, are today known as Bernstein’s conditions. (These conditions are the main result of [3], whereas they are a mere corollary in [7].) The three conditions are known, today, as the absence of true dependencies, of anti-dependencies, and of output dependencies, and appear in this form in standard texts (see, for example, [2]).

Bernstein considers fetches and stores, whereas input and output regions are more general than this. For example, if $x$ is saved and restored by $f$, then $x$ is not part of the output region of the effect of $f$, even though store operations are made to $x$. Similarly, if $x$ is a properly initialized temporary variable of $f$, then $x$ is not part of the input region of $f$, even though fetch operations are made from $x$. Both of these facts might help in proving that $f$ can be interchanged with an adjacent section $g$. Note that this does not necessarily mean that $f$ and $g$ can be done in parallel (which was the main point of [3]), since the usages of $x$ by $f$ and by $g$ can easily interfere with one another.

The results obtained here are capable of extension in several directions, so as
to be applicable to a more general program. In particular:

- Theorem 5.3 may be iterated, as we have seen in the proof of Theorem 6.1. That is, a given program may be contracted and expanded, then contracted and expanded further, and so on, with the effect and its guard remaining unchanged. This has implications for top-down and bottom-up design, as well as for structured programs.

- Theorem 5.3 may be generalized to the case of non-contiguous sections. This has implications for structured programs which use constructs such as `break` and `continue` in C-like languages, or `redo` in Perl.

- The results of this paper may be extended to handle expressions with side effects. This includes differences between left-to-right and right-to-left evaluation; assignments with left sides having side effects; and exceptions and their equivalents.

- The ideas of commutativity and simple section interchange may be generalized to commutativity except on temporary variables, continuing a theme first introduced by Igarashi (see de Bakker [5]).

- A far more difficult problem is the establishment of a necessary, as well as a sufficient, condition for two sections of a program to be interchangeable; that is, for their effects to commute. This becomes a problem in group theory, since effects, as functions \( e : S \to S \), form a non-abelian group \( G \) under composition of functions. Even the determination of those effects which commute with *every* other effect becomes a matter of finding the center of \( G \), which is itself a subgroup of \( G \).

References


