A Comparative Study of Invariant Assertions, Invariant Relations, and Invariant Functions

Asma Louhichi
FST, Univ Tunis El Manar
Tunis, Tunisia
louhichiasma@yahoo.fr
, Olfa Mraihi
ISG
bardo, Tunisia
olfa.mraihi@yahoo.fr
, Lamia Labed Jilani
ISG
Bardo, Tunisia
lamia.labed@planet.tn
and Ali Mili
NJIT
Newark, NJ USA
mili@cis.njit.edu

Abstract

Invariant assertions play an important role in the analysis and documentation of while loops of imperative programs. Invariant functions and invariant relations are alternative analysis tools that are distinct from invariant assertions but are related to them. In this paper we discuss these three concepts and analyze their relationships. The study of invariant functions and invariant relations is interesting not only because it provides alternative means to analyze loops, but also because it gives us insights into the structure of invariant assertions, and may help us enhance techniques for generating invariant assertions.

Keywords

Invariant assertions, invariant functions, invariant relations, loop invariants, program analysis, program verification, while loops.

1 Introduction

In [5], Hoare introduced the concept of an *invariant assertion* as a useful tool in the analysis of while loops. The study of invariant assertions, and their use in the analysis of while loops, have been the focus of active research in the seventies and eighties, and the subject of renewed interest in the last few years. In this paper we wish to investigate the relationships between this well known, thoroughly researched, concept, and two distinct but related concepts: invariant functions [12] and invariant relations [9]. We consider a while loop w on space S defined by $w =$ `while t do B`. These three concepts can be summarily and informally characterized as follows:

- An invariant assertion (as defined for the purposes of our discussion) is a predicate α on S that is preserved by application of the loop body.
- An invariant function is a total function on S whose value is preserved by application of the loop body.
- An invariant relation is a reflexive transitive relation that is a superset of the loop body's function.

In section 3, we give definitions of all three concepts, using a uniform model, namely the calculus of relations, which we introduce in section 2. In section 4 we discuss the interrelations between these concepts, and explore complementarities between their respective insights on loop behavior. In section 5 we summarize our results and explore venues of further research.

2 Relational Mathematics

2.1 Elements of Relations

We consider a set S defined by the values of some program variables, say x, y and z; we typically refer to elements of S by s, and we note that s has the form $s = \langle x, y, z\rangle$. We use the notation $x(s)$, $y(s)$, $z(s)$ to refer to the x-component, y-component and z-component of s. We may sometimes use x to refer to x(s) and x' to refer to $x(s')$, when this raises no ambiguity. We use relations to represent program specifications and program functions; we then refer to S as the *space* of the specification, and also refer to it as the space of any program that manipulates these variables. Constant relations on some set S include the *universal* relation, denoted by L, the *identity* relation, denoted by I, and the *empty* relation, denoted by $\emptyset$. Given a predicate t, we denote by $I(t)$ the subset of the identity relation defined as follows: $I(t) = \{(s, s')|s' = s \wedge t(s)\}$.

Because relations are sets, we use the usual set theoretic operations between relations (union, intersection, complement, cartesian product). Operations on relations also include the *converse*, denoted by $\widehat{R}$, and defined by $\widehat{R} = \{(s, s')|(s', s) \in R\}$. The *product* of relations R and R' is the relation denoted by $R \circ R'$ (or RR') and defined by $R \circ R' = \{(s, s')|\exists t : (s, t) \in R \wedge (t, s') \in R'\}$. The *prerestriction* (resp. *post-restriction*) of relation R to predicate t is the relation $\{(s, s')|t(s) \wedge (s, s') \in R\}$ (resp. $\{(s, s')|(s, s') \in R \wedge t(s')\}$). We admit without proof that the pre-restriction of a relation R to predicate t is $I(t) \circ R$ and the post-restriction of relation R to predicate t is $R \circ I(t)$. The *domain* of relation R is defined as $dom(R) = \{s|\exists s' : (s, s') \in R\}$. The *range* of relation R is denoted by $rng(R)$ and defined as $dom(\widehat{R})$. We admit without proof that for a relation R, $RL = \{(s, s')|s \in dom(R)\}$ and $LR = \{(s, s')|s' \in rng(R)\}$. The *nucleus* of relation R is the relation denoted by $\mu(R)$ and defined as $R\widehat{R}$.

We say that R is *deterministic* (or that it is a *function*) if and only if $\widehat{R}R \subseteq I$, and we say that R is *total* if and only if $I \subseteq R\widehat{R}$, or equivalently, $RL = L$. Given two total deterministic relations R and R', we say that R is *more-injective* than R' if and only if $R\widehat{R} \subseteq R'\widehat{R'}$. To understand this definition, consider that each function partitions its domain into equivalence classes, called the *level sets* of the function [6]; a more-injective function is one whose level sets define a finer partition of the domain. A total deterministic relation R is said to be *injective* if and only if it is more-injective than I.

A relation R is said to be *rectangular* if and only if $R = RLR$. A relation R is said to be *reflexive* if and only if $I \subseteq R$, *transitive* if and only if $RR \subseteq R$ and *symmetric* if and only if $R = \widehat{R}$. We are interested in two special types of rectangular relations: rectangular surjective relations are called *vectors* and satisfy the condition $RL = R$; rectangular total relations are called *invectors* (inverse of a vector) and satisfy the condition $LR = R$. In set theoretic terms, a vector on set S has the form $A \times S$, and an invector has the form $S \times A$, for some subset A of S. Vector $A \times S$ can also be written as $I(A) \circ L$.

2.2 Refinement Ordering

We define an ordering relation on relational specifications under the name *refinement ordering*:

Definition 1. *A relation R is said to* refine *a relation R' if and only if*

$$RL \cap R'L \cap (R \cup R') = R'.$$

In set theoretic terms, this equation means that the domain of R is a superset of (or equal to) the domain of R', and that for each element in the domain of R', the set of images by R is a subset of (or equal to) the set of images by R'. This is similar to, but different from, refining a pre/postcondition specification by weakening its precondition and/or strengthening its postcondition [4, 14]. We abbreviate this property by $R \sqsupseteq R'$ or $R' \sqsubseteq R$. We admit that, modulo traditional definitions of total correctness [3, 4, 7], the following propositions hold:

- A program P is correct with respect to a specification R if and only if $[P] \sqsupseteq R$, where $[P]$ is the function defined by P (we may, by abuse of notation use P to refer to $[P]$ when the context allows).
- $R \sqsupseteq R'$ if and only if any program correct with respect to R is correct with respect to R'.

Intuitively, R refines R' if and only if R represents a stronger requirement than R'.

2.3 Refinement Lattice

We admit without proof that the refinement relation is a partial ordering. In [2] Boudriga et al. analyze the lattice properties of this ordering and find the following results:

- Any two relations R and R' have a greatest lower bound, which we refer to as the *meet*, denote by $\sqcap$, and define by:
$$R \sqcap R' = RL \cap R'L \cap (R \cup R').$$
- Two relations R and R' have a least upper bound if and only if they satisfy the following condition (which we refer to as the *consistency condition*):
$$RL \cap R'L = (R \cap R')L.$$
Under this condition, their least upper bound is referred to as the *join*, denoted by $\sqcup$, and defined by:
$$R \sqcup R' = \overline{RL} \cap R' \cup \overline{R'L} \cap R \cup (R \cap R').$$
Intuitively, the join of R and R', when it exists, behaves like R outside the domain of R', behaves like R' outside the domain of R, and behaves like the intersection of R and R' on the intersection of their domain. The consistency condition provides that the domain of their intersection is identical to the intersection of their domains.
- Two relations R and R' have a least upper bound if and only if they have an upper bound; this property holds in general for lattices, but because the refinement ordering is not a lattice (since the existence of the join is conditional), it bears checking for this ordering specifically.
- The lattice of refinement admits a *universal lower bound*, which is the empty relation.
- The lattice of refinement admits no *universal upper bound.*
- Maximal elements of this lattice are total deterministic relations.

See Figure 1. The outline of this figure shows the overall structure of the lattice of specifications.

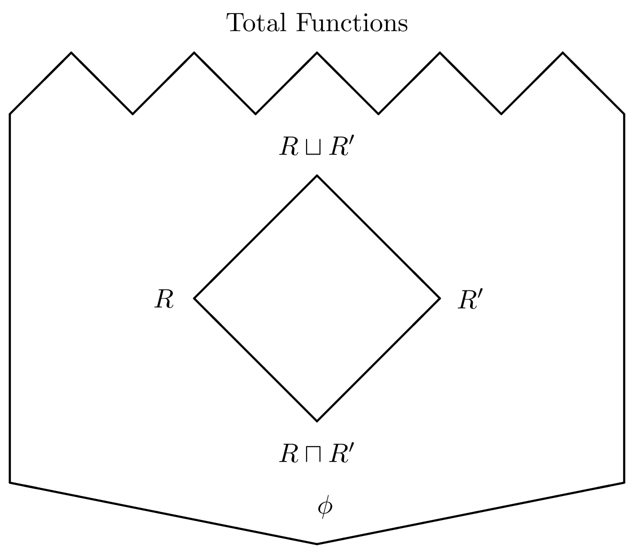

Figure 1: Lattice Structure of Refinement

2.4 Loop Semantics

Our study requires that we present some background on loop semantics, which we use throughout the paper to support our discussions and our arguments. Because these are not original results, we present them briefly, without proof. We consider a deterministic program P on some variables x_1, x_2, ... x_n. We let S be the space defined by all the values that the aggregate of variables may take. Because P is deterministic, its semantics is captured by a function, which we denoted by $[P]$ and define by

$$[P] = \{(s, s')| \text{ if } p \text{ starts execution in state } s \text{ then it terminates in state } s'\}.$$

From this definition, it stems that $dom([P])$ can be interpreted as

$$dom([P]) = \{(s, s')| \text{ if } P \text{ starts execution in state } s \text{ then it terminates}\}.$$

We submit two fundamental theorems about loops, which we will use subsequently.

Theorem 1. Sub-goal Induction Theorem. *We consider a while loop $w =$ `while t do B` on space S and a total relation R on the same space S (such that $dom(R) = S$), w is correct with respect to R if:*

- $dom([w]) = S$.
- $I(\neg t) \subseteq R$.
- $I(t) \circ [B] \circ R \subseteq R$.

This theorem is due to Morris and Wegbreit [15]; we have rewritten it in relational terms, to make it useful for our purposes.

Theorem 2. Mills Theorem. *We consider a while loop $w =$ `while t do B` on space S that terminates for all states in S, and a function W on the same space S, then $[w] = W$ if and only if:*

- $dom(W) = S$.
- $I(\neg t) \circ W = I(\neg t)$.
- $I(t) \circ [B] \circ W = I(t) \circ W$.

This theorem is due to H.D. Mills [13]. Even though it was derived independently from (and prior to) the Sub-goal Induction Theorem, it could be considered as a special case of it (the case where the specification at hand, R, is deterministic). Because these two theorems assume that $[w]$ is total, we are interested in restricting our study to loops that meet this condition; the proposition below provides that this assumption causes no loss of generality.

Proposition 1. *We consider a while loop w on space S. We let s_0 be an element of $dom([w])$ and we denote by s_1, s_2, ... s_n be the sequence of states obtained by the successive executions of the loop body, where s_n is the state obtained when the loop terminates. Then, for all i, $0 \leq i \leq n$, $s_i \in dom([w])$.*

Since initial states, intermediate states, and final states are all in $dom([w])$, we can let S be $dom([w])$ without loss of generality, as then all the states of interest are within S. This choice of state space makes the while statement's function total by construction. In the sequel, we implicitly assume this condition throughout.

The following theorem represents, for our purposes, our semantic definition of while loops.

Theorem 3. *Given a while statement of the form $w =$ `while t do B`. If w terminates for all the states in S, then $[w] = (I(t) \circ [B])^* I(\neg t)$.*

3 Invariants

In this section we present in turn the three invariants in parallel and present, for each: a formal definition in relational terms, an ordering relation between invariants, a characterization of the strongest invariant, and an elucidation of their relation to the function of the loop.

3.1 Formal Definitions

We consider a while statement of the form $w =$ `while t do B` on some space S, and we assume that w terminates normally for any initial state s in S. We define in turn, invariant assertions, invariant relations, and invariant functions. For the sake of uniformity, we use the same mathematical baggage to represent them, namely the calculus of relations.

For the sake of illustration, we consider a running example on which we apply our definitions and propositions. We consider the following while loop on array variables a and b, real variables x and y, and index variables i and j:

```
while (i!=N+1)
{x=x+a[i]; y=y+b[j]; i=i+1; j=j-1;}
```

3.1.1 Invariant Assertions

Traditionally, an invariant assertion α for the while loop $w =$ `while t do B` with respect to a precondition/ postcondition pair (ϕ, ψ) is defined as a predicate on S that satisfies the following conditions:

- $\phi \Rightarrow \alpha$.
- $\{\alpha \wedge t\}B\{\alpha\}$.
- $\alpha \wedge \neg t \Rightarrow \psi$.

As defined, the invariant assertion is dependent not only on the while loop, but also on the loop's specification, in the form of a precondition/ postcondition pair. This precludes meaningful comparisons with invariant relations and invariant functions, which are dependent solely on the loop. Hence we redefine the concept of invariant assertion in terms of the second condition alone. Also, to represent an invariant assertion, we map the predicate on S into a vector (a relation) on S. Specifically, we represent the predicate α by the vector A defined by

$$A = \{(s, s')|\alpha(s)\}.$$

Definition 2. *Given a while statement of the form,* `w = while t do B`*, an* invariant assertion *is defined as a non-empty vector A on S that satisfies the following condition:*

$$A \cap T \cap [B] \subseteq \widehat{A},$$

where T is the vector defined by predicate t, i.e. $T = \{(s, s')|t(s)\}$.

This is a straightforward interpretation, in relational terms, of the second condition (above), $\{\alpha \wedge t\}B\{\alpha\}$. For the sake of illustration, we submit that the following vector satisfies the condition of Definition 2: $A = \{(s, s')|x = \sum_{k=1}^{i-1} a[k]\}$. We compute the left hand side of the definition:

$$\begin{aligned}
& A \cap T \cap [B] \\
= \quad & \{ \text{ Substitutions } \} \\
& \{(s, s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge i \neq N+1 \wedge a' = a \wedge b' = b \wedge \\
& x' = x + a[i] \wedge i' = i+1 \wedge y' = y + b[j] \wedge j' = j-1\} \\
\subseteq \quad & \{ \text{ Set Theory } \} \\
& \{(s, s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge i \neq N+1 \wedge a' = a \wedge x' = x + a[i] \wedge i' = i+1\} \\
= \quad & \{ \text{ Simplification } \} \\
& \{(s, s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge i \neq N+1 \wedge a' = a \wedge x' = \sum_{k=1}^{i-1} a[k] + a[i] \wedge i' = i+1\} \\
= \quad & \{ \text{ Simplification } \} \\
& \{(s, s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge i \neq N+1 \wedge a' = a \wedge x' = \sum_{k=1}^{i} a[k] \wedge i' = i+1\} \\
= \quad & \{ \text{ Substitution } \} \\
& \{(s, s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge i \neq N+1 \wedge a' = a \wedge x' = \sum_{k=1}^{i'-1} a'[k] \wedge i' = i+1\} \\
\subseteq \quad & \{ \text{ Set Theory } \} \\
& \{(s, s')|x' = \textstyle\sum_{k=1}^{i'-1} a'[k]\} \\
= \quad & \{ \text{ Substitution } \} \\
& \widehat{A}.
\end{aligned}$$

Note that we have not proven that the assertion $x = \sum_{k=1}^{i-1} a[k]$ holds after each iteration; rather we have only proven that if this assertion holds at one iteration, then it holds at the next iteration. This is in effect an inductive proof without a basis of induction.

3.1.2 Invariant Relations

Invariant relations are reflexive transitive relations that are supersets of the loop body's function. By virtue of this definition, one can see that an invariant relation contains pairs of states (s, s') such that s' follows from s by application of an arbitrary number (including zero) of iterations.

Definition 3. *Given a while loop of the form* `w = while t do B` *on some space S, and given a relation R on S, we say that R is an* invariant relation *for w if and only if R is reflexive, transitive, and satisfies the following conditions (where T is the vector defined by predicate t and $\overline{T}$ is the complement of T):*

- *The Invariance Condition: $T \cap [B] \subseteq R$.*
- *The Convergence Condition: $R \circ \overline{T} = L$.*

To highlight its important properties, we may sometimes refer to an invariant relation as a *reflexive transitive invariant relation*; these two terms refer to the same concept. Note that unlike the definition of invariant assertions (Definition 2) the definition of invariant relations (Definition 3) is not recursive: the term R appears on one side only of each equation. What makes it inductive, nevertheless, is the fact that R is reflexive and transitive; reflexivity serves the basis of induction, and transitivity serves the inductive step of an inducytive proof to the effect that this relation holds between any two states s and s' such that s' is obtained from s by application of an arbitrary number of iterations.

As for the *convergence condition*, it provides that any state in S can be mapped by R onto a state in S that satisfies $\neg t$. Given that R represents the effect of applying the loop body an arbitrary number of times, this condition ensures that these applications eventually produce a final state, i.e. a state that causes the loop to terminate.

To illustrate this concept, we consider again the example of the array sum, presented above, and propose the following invariant relation for it.

$$R = \{(s, s')|x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge i \leq i'\}.$$

This relation is clearly reflexive and transitive. To check the invariance condition, we compute the intersection $T \cap [B] \cap R$ and check that it equals $T \cap [B]$.

$T \cap [B] \cap R$

$=$ { Substitution }

$\{(s, s')|i \neq N + 1 \wedge x' = x + a[i] \wedge i' = i + 1 \wedge a' = a$
$\wedge y' = y + b[j] \wedge j' = j - 1 \wedge b' = b \wedge x + \sum_{k=i}^{N} a[k]$
$= x' + \sum_{k=i'}^{N} a'[k] \wedge i \leq i'\}$

$=$ { Substitution }

$\{(s, s')|i \neq N + 1 \wedge x' = x + a[i] \wedge i' = i + 1 \wedge a' = a$
$\wedge y' = y + b[j] \wedge j' = j - 1 \wedge b' = b \wedge x + \sum_{k=i}^{N} a[k]$
$= x + a[i] + \sum_{k=i+1}^{N} a'[k] \wedge i \leq i'\}$

$=$ { Simplification }

$\{(s, s')|i \neq N + 1 \wedge x' = x + a[i] \wedge i' = i + 1 \wedge a' = a$
$\wedge y' = y + b[j] \wedge j' = j - 1 \wedge b' = b \wedge x + \sum_{k=i}^{N} a[k]$
$= x + \sum_{k=i}^{N} a'[k] \wedge i \leq i'\}$

$=$ { Logical Simplification }

$\{(s,s')|i \neq N+1 \wedge x' = x + a[i] \wedge i' = i+1 \wedge a' = a$
$\wedge y' = y + b[j] \wedge j' = j-1 \wedge b' = b \wedge i \leq i'\}$
$=$ { Associativity }
$\{(s,s')|i \neq N+1 \wedge x' = x + a[i] \wedge (i' = i+1 \wedge i \leq i')$
$\wedge a' = a \wedge y' = y + b[j] \wedge j' = j-1 \wedge b' = b\}$
$=$ { Logical Simplification }
$\{(s,s')|i \neq N+1 \wedge x' = x + a[i] \wedge i' = i+1$
$\wedge a' = a \wedge y' = y + b[j] \wedge j' = j-1 \wedge b' = b\}$
$=$ { Substitution }
$T \cap [B]$.

To check the convergence condition, we compte $R \circ \overline{T}$ and check that it is the full relation.

$R \circ \overline{T}$
$=$ { Substitution }
$\{(s,s')|x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge i \leq i'\} \circ \{(s,s')|i = N+1\}$
$=$ { Relational Product }
$\{(s,s')|\exists s'' : x + \sum_{k=i}^{N} a[k] = x'' + \sum_{k=i''}^{N} a''[k] \wedge i \leq i'' \wedge i'' = N+1\}$
$=$ { Simplification }
$\{(s,s')|\exists s'' : x + \sum_{k=i}^{N} a[k] = x'' \wedge i \leq N+1 \wedge i'' = N+1\}$
$=$ { Simplification/ Interpretation }
$\{(s,s')|\mathbf{true}\}$
$=$ { Definition }
L.

3.1.3 Invariant Functions

Invariant functions are functions whose value remains unchanged by application of the loop body's function [12].

Definition 4. *Let w be a while statement of the form* `while t do B` *that terminates normally for all initial states in S. We say that a function F on S is an* invariant function *if and only if it is total and satisfies the following condition: $(T \cap [B]) \circ F = T \cap F$.*

To illustrate the concept of invariant function, we consider the array program, and submit the following function:

$$F\begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix} = \begin{pmatrix} a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \\ a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \end{pmatrix}.$$

We briefly verify that this function is invariant with respect to application of the loop body (assuming s satisfies condition t).

$$F\left([B]\begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix}\right) = F\begin{pmatrix} a \\ x + a[i] \\ i+1 \\ b \\ y + b[j] \\ j-1 \end{pmatrix} = \begin{pmatrix} a \\ x + a[i] + \sum_{k=i+1}^{N} a[k] \\ N+1 \\ a \\ x + a[i] + \sum_{k=i+1}^{N} a[k] \\ N+1 \end{pmatrix} = \begin{pmatrix} a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \\ a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \end{pmatrix}$$

$$= F \begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix}.$$

3.2 Orderings Invariants

Invariants are not created equal. For example **true** is an invariant assertion for any loop, though not a very useful assertion; the full relation (L) is an invariant relation for any loop, though not a very useful relation; finally a constant function is an invariant function for any loop, though not a very useful function. In this section we briefly review how each type of invariant is ordered.

3.2.1 Ordering Invariant Assertions by Implication

Invariant assertions are naturally ordered by logical implication. As we recall from Definition 2, the invariant assertions are represented by non-empty vectors; hence stronger invariant assertions are represented by smaller vectors. Even though $\alpha =$ **false** does satisfy the condition

$$\{\alpha \wedge t\}B\{\alpha\},$$

we do not consider it as a legitimate invariant assertion because it cannot be represented by a non-empty vector. The inclusion ordering among non empty vectors defines a lattice structure, where the join is the intersection of vectors and the meet is the union of vectors.

For the sake of illustration, we consider the following invariant assertions, say A_0 and A_1, defined as: $A_0 = \{(s,s')|x = \sum_{k=1}^{i-1} a[k]\}, A_1 = \{(s,s')|x = \sum_{k=1}^{i-1} a[k] \wedge y = \sum_{k=j+1}^{N} b[k]\}$. To check that A_1 is an invariant assertion of the loop, we compute the expression $A_1 \cap T \cap [B]$. We find,

$$\begin{aligned}
& A_1 \cap T \cap [B] \\
= \quad & \{ \text{ Substitutions } \} \\
& \{(s,s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge y = \sum_{k=j+1}^{N} b[k] \wedge i \neq N+1 \wedge a' = a \wedge b' = b \wedge \\
& x' = x + a[i] \wedge i' = i+1 \wedge y' = y + b[j] \wedge j' = j-1\} \\
\subseteq \quad & \{ \text{ Simplification, Substitution } \} \\
& \{(s,s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge y = \sum_{k=j+1}^{N} b[k] \wedge a' = a \wedge b' = b \wedge \\
& x' = \textstyle\sum_{k=1}^{i} a[k] \wedge i' = i+1 \wedge y' = \sum_{k=j}^{N} b[k] \wedge j' = j-1\} \\
= \quad & \{ \text{ Substituting Indices } \} \\
& \{(s,s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge y = \sum_{k=j+1}^{N} b[k] \wedge a' = a \wedge b' = b \wedge \\
& x' = \textstyle\sum_{k=1}^{i'-1} a[k] \wedge i' = i+1 \wedge y' = \sum_{k=j'+1}^{N} b[k] \wedge j' = j-1\} \\
= \quad & \{ \text{ Substituting Arrays } \} \\
& \{(s,s')|x = \textstyle\sum_{k=1}^{i-1} a[k] \wedge y = \sum_{k=j+1}^{N} b[k] \wedge a' = a \wedge b' = b \wedge \\
& x' = \textstyle\sum_{k=1}^{i'-1} a'[k] \wedge i' = i+1 \wedge y' = \sum_{k=j'+1}^{N} b'[k] \wedge j' = j-1\} \\
\subseteq \quad & \{ \text{ Set Theory } \} \\
& \{(s,s')| \qquad x' = \textstyle\sum_{k=1}^{i'-1} a'[k] \wedge y' = \sum_{k=j'+1}^{N} b'[k]\} \\
= \quad & \{ \text{ Substitution } \} \\
& \widehat{A_1}.
\end{aligned}$$

We leave it to the reader to check that A_0 is also an invariant assertion. Also, because A_1 is a subset of A_0, we find that A_1 represents a stronger assertion of the loop.

3.2.2 Ordering Invariant Relations by Refinement

Invariant relations are ordered by refinement, which, as we have discussed in section 2.2, has lattice-like properties. More refined relations give more information on loop behavior. Because they are by definition reflexive, invariant relations are total. If we consider the definition of refinement (Definition 1), we find that it can be simplified as follows

$RL \cap R'L \cap (R \cup R') = R'$

$\Leftrightarrow$ $\{$ R and R' are total $\}$

$L \cap L \cap (R \cup R') = R'$

$\Leftrightarrow$ $\{$ Set Theory $\}$

$R \cup R' = R$

$\Leftrightarrow$ $\{$ Set Theory $\}$

$R \subseteq R'$.

Hence for invariant relations, refinement is synonymous with set inclusion. We illustrate this ordering with a simple example. We leave it to the reader to check that the following two relations are invariant relations for the array sum program.

$$R_0 = \{(s,s')|a = a' \wedge x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k]\}.$$

$$R_1 =$$

$$\{(s,s')|a = a' \wedge x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge b = b' \wedge y + \sum_{k=1}^{j} b[k] = y' + \sum_{k=1}^{j'} b'[k] \wedge i + j = i' + j'\}.$$

Invariant relation R_1 refines invariant relation R_0 because they have the same domain and the latter is a subset of the former. Clearly, the latter also provides more information on loop behavior than the former.

3.2.3 Ordering Invariant Functions by Injectivity

Invariant functions are total by definition, hence they all have the same domain. When we write an equation such as $F(s) = F([B](s))$, this gives all the more information on B that F is more injective, i.e. partitions its domain finely. If F were a constant, then this equation does not tell us anything about B, since it holds for all B. Hence we order invariant functions by injectivity.

To illustrate this ordering, we consider the following two invariant functions, and we leave it to the reader to check that they are indeed invariant functions for the array sum program:

$$F_0 \begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix} = \begin{pmatrix} a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \\ a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \end{pmatrix} .F_1 \begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix} = \begin{pmatrix} a \\ x + \sum_{k=i}^{N} a[k] \\ N+1 \\ b \\ y + \sum_{k=1}^{j} b[k] \\ i + j - N - 1 \end{pmatrix}.$$

To check which (if any) of F_0 and F_1 is more-injective than the other, we compute their nuclei, and find

$$\mu(F_0) = \{(s,s')|a = a' \wedge x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k]\}.$$

$$\mu(F_1) = \{(s, s')|a = a' \wedge x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge$$

$$b = b' \wedge y + \sum_{k=1}^{j} b[k] = y' + \sum_{k=1}^{j'} b'[k] \wedge i + j = i' + j'\}.$$

Clearly, F_1 is more-injective than F_0, since $\mu(F_1) \subseteq \mu(F_0)$.

3.3 Invariants and the Loop Function

In this section we present theorems that relate the invariant assertions, invariant relations and invariant functions to the function of the loop.

3.3.1 Invariant Assertions and Loop Functions

In this section we consider a theorem that provides an interesting link between the function of the loop and an adequate (in a sense to be defined) invariant assertion. We consider the following program structure, which we annotate by intermediate assertions and invariant assertions:

```
f =
begin {s=s0} init; {s=[init](s0)}
while t do  {F(s)=F(s0) && s in dom(W inter F)} B;
{s=F(s0)}
end.
```

A theorem by Mili et al. [8], which is based on earlier findings by Morris and Wegbreit [15], Mills [13] and Basu and Misra [1] provides that program f computes some total function F on S if:

1. The specification Y defined by $Y = F\widehat{F} \cap L(\widehat{F \cap W})$ (where W is the function of the while loop) is total.

2. Segment `init` is correct with respect to specification $Y = F\widehat{F} \cap L(\widehat{F \cap W})$.

3. The following predicate is an invariant assertion: $\alpha(s_0, s) \equiv (s_0, s) \in F\widehat{F} \cap L(\widehat{F \cap W})$.

The invariant assertion α is adequate, in the sense that it is strong enough that, if the loop does compute function F, then this loop invariant allows us to construct a Hoare-style proof to this effect.

3.3.2 Invariant Relations and Loop Function

In this section, we submit two propositions that elucidate the relation between invariant relations and loop functions. The first proposition shows us how to derive an invariant relation from the function of the loop.

Proposition 2. *Given a while loop w on space S of the form* `w: while t do B`*; we assume that w terminates for all s in S, and we let W be the function defined by w. Then $R = \mu(W)$ is an invariant relation for w.*

Proof. We establish in turn the invariance condition, then the convergence condition of Definition 3. But First, we briefly check that R is reflexive and transitive. Relation R is reflexive by virtue of being the nucleus of a total relation; also, it is transitive by virtue of the following brief argument:

$$\begin{array}{ll}
 & R \circ R \\
= & \quad \{ \text{ Substitution, definition of nucleus } \} \\
 & (W\widehat{W})(W\widehat{W}) \\
\subseteq = & \quad \{ \text{ Associativity, determinacy of } W \ \} \\
 & W\widehat{W} \\
= & \quad \{ \text{ substitution } \} \\
 & R.
\end{array}$$

The Invariance Condition. We consider the third condition of Mills' Theorem (Theorem 2), and we proced by logical implication.

$$\begin{array}{ll}
 & I(t) \circ [B] \circ W = I(t) \circ W \\
\Leftrightarrow & \quad \{ \text{ Monotonicity } \} \\
 & I(t) \circ [B] \circ W \subseteq W \\
\Leftrightarrow & \quad \{ \text{ Multiplying on both sides by } \widehat{W}\text{, Monotonicity } \} \\
 & I(t) \circ [B] \circ W \circ \widehat{W} \subseteq W \circ \widehat{W} \\
\Leftrightarrow & \quad \{ \text{ Substitution } \} \\
 & I(t) \circ [B] \circ R \subseteq R \\
\Leftrightarrow & \quad \{ \text{ Reflexivity of } R \ \} \\
 & I(t) \circ [B] \subseteq R \\
\Leftrightarrow & \quad \{ \text{ Rewriting } \} \\
 & T \cap [B] \subseteq R.
\end{array}$$

The Convergence Condition. We consider the left hand side of the convergence condition in Definition 3, and proceed as follows:

$$\begin{array}{ll}
 & R \circ \overline{T} \\
= & \quad \{ \text{ Substitution, rewriting } \} \\
 & W \circ \widehat{W} \circ I(\neg t) \circ L \\
= & \quad \{ \text{ Laws of inverse, associativity } \} \\
 & W \circ (\widehat{I(\neg t) \circ W}) \circ L \\
= & \quad \{ \text{ Second condition, Mills Theorem } \} \\
 & W \circ I(\neg t) \circ L \\
= & \quad \{ \text{ Post restriction of } W \text{ to its range } \} \\
 & W \circ L \\
= & \quad \{ \text{ Totality of } W \ \} \\
 & L.
\end{array}$$

qed

Proposition 2 shows us how to derive an invariant relation from the loop function; a much more useful result in practice is how to derive the function of the loop (or an approximation thereof) from an invariant relation. This is the subject of the following proposition.

Proposition 3. *We consider a while loop w on space S of the form* `w: while t do B`*, which* *terminates for any element in S. If R is an invariant relation of w then*

$$[w] \sqsupseteq R \cap \overline{T}.$$

The proof of this Proposition is given in [10], hence will not be given here. Given that this proposition provides a lower bound of the loop function, one may want to know under what condition the lower bound is tight enough to be the actual loop function. The answer is fairly straightforward:

- The relation $R = (T \cap [B])^*$ is an invariant relation, since it is reflexive and transitive, by construction. Also, it is a superset of $(T \cap [B])$, by construction, and it satisfies the convergence condition by virtue of the totality of W.
- The relation $R = (T \cap [B])^*$ is the *smallest* invariant relation of the loop. Indeed, the reflexive transitive closure of $(T \cap [B])$ is the smallest relation that is a superset of $(T \cap [B])$.
- The lower bound of $[w]$ obtained from this invariant relation is the function of the loop, by virtue of Theorem 3.

This concludes the discussion of the relation between invariant relations of a loop and the function of the loop.

3.3.3 Invariant Functions and Loop Function

A theorem by Mili et al. [12] provides that the function of the loop is an invariant function of the loop.

4 Relations between Invariants

4.1 Invariant Assertions and Invariant Relations

We have shown in [11] that from an invariant relation we can infer an invariant assertion. We have not found a way to infer an invariant relation from an invariant assertion, because invariant assertions are unary relations whereas invariant relations are binary relations.

4.2 Invariant Relations and Invariant Functions

In [11] we have shown that

- From an invariant function F, we can derive an invariant relation R as the nucleus of F. Such invariant relations are symmetric, in addition to being reflexive and transitive.
- Any invariant relation that is symmetric, in addition to being reflexive and transitive, is the nucleus of an invariant function. For example, the following invariant relation

$$R = \{(s, s')|x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge a = a'\}.$$

is the nucleus of the invariant function,

$$F\begin{pmatrix} a \\ x \\ i \\ b \\ y \\ j \end{pmatrix} = \begin{pmatrix} a \\ x+\sum_{k=i}^{N} a[k] \\ N+1 \\ a \\ x+\sum_{k=i}^{N} a[k] \\ N+1 \end{pmatrix}.$$

whereas the following invariant relation

$$R = \{(s,s')|x+\sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k] \wedge i \leq i'\}$$

is not the nucleus of any invariant function because it is not symmetric.

Contrary to what the names suggest, an invariant function is not an invariant relation that happens to be deterministic. The only relation that is reflexive and deterministic at once is the identity, and the only loop body that accepts the identity as an invariant relation is `skip`, which is of no interest.

4.3 Invariant Assertions and Invariant Functions

Contrary to what the names may suggest, an invariant assertion is not an invariant function that takes boolean values. Rather an invariant function takes the same values before and after application of the loop body whereas an invariant assertion may be false before and become true after. This seems to indicate that a more appropriate name for invariant assertions is *monotonic assertions*. Hence we may distinguish between *monotonic assertions*, which satisfy

$$\{\alpha \wedge t\}B\{\alpha\}$$

(traditionally called invariant assertions) and genuinely invariant assertions, which satisfy

$$\{\alpha \wedge t\}B\{\alpha\}$$

$$\{\neg\alpha \wedge t\}B\{\neg\alpha\}.$$

This distinction is not a mere exercise in hair-splitting. We conjecture (but have not yet proven) that an invariant assertion is usually made up of an invariant part and a monotonic part; and that the invariant part can be derived from invariant functions. Because we have some machinery to derive invariant functions by static code analysis, this may be useful to derive the invariant part of invariant assertions.

5 Brief Concluding Remarks

This is an unfinished work; we intend to explore in detail the characteristics of three types of invariants, and investigate their relations, but there is a lot more to say about these concepts than we could fit in this this paper. We view this line of research as having two broad goals: first broaden the scope of tools that we use to analyze while loops; second, deploy the insights gained from invariant relations and invariant functions to enhance the study of invariant assertions.

Despite the tentative nature of our preliminary conclusions, we are confident of two premises:

- Given an invariant assertion, we cannot use it to derive an invariant relation.
- Given an invariant generation method, we cannot use it to derive an invariant relation generation method.

Because invariant functions and invariant relations appear to be useful in generating invariant assertions, we are exploring means to refine our invariant function and invariant relation generation techniques; we are also interested in analyzing the structure of invariant assertions, to see which parts can be derived by means of invariant assertions and invariant functions.

References

[1] S.K. Basu and J.D. Misra. Proving loop programs. *IEEE Transactions on Software Engineering*, 1(1):76–86, 1975.

[2] N. Boudriga, F. Elloumi, and A. Mili. The lattice of specifications: Applications to a specification methodology. *Formal Aspects of Computing*, 4:544–571, 1992.

[3] E.W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[4] D. Gries. *The Science of programming.* Springer Verlag, 1981.

[5] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576 – 583, October 1969.

[6] R.C. Linger, H.D. Mills, and B.I. Witt. *Structured programming.* Addison Wesley, 1979.

[7] Z. Manna. *A Mathematical Theory of Computation.* McGraw Hill, 1974.

[8] A. Mili, J. Desharnais, and F. Mili. Relational heuristics for the design of deterministic programs. *Acta Informatica*, 24(3):239–276, 1987.

[9] Ali Mili. Reflexive transitive loop invariants: A basis for computing loop functions. In *First International Workshop on Invariant Generation*, Hagenberg, Austria, June 2007.

[10] Ali Mili, Shir Aharon, and Chaitanya Nadkarni. Mathematics for reasoning about loop functions. Technical report, New Jersey Institute of Technology, web.njit.edu/m̃ili/fxscp.pdf, 2009.

[11] Ali Mili, Shir Aharon, Chaitanya Nadkarni, Olfa Mraihi, Asma Louhichi, and Lamia Labed Jilani. Reflexive transitive invariant relations: A basis for computing loop functions. *Journal of Symbolic Computation*, 45:1114–1143, 2009.

[12] Ali Mili, Jules Desharnais, and Jean Raymond Gagne. Strongest invariant functions: Their use in the systematic analysis of while statements. *Acta Informatica*, April 1985.

[13] H.D. Mills. The new math of computer programming. *Communications of the ACM*, 18(1), January 1975.

[14] C.C. Morgan. *Programming from Specifications.* International Series in Computer Sciences. Prentice Hall, London, UK, 1998.

[15] J.H. Morris and B. Wegbreit. Program verification by subgoal induction. In R.T. Yeh, editor, *Current Trends in Programming Methodology*, volume II, chapter 8. Prentice Hall, Englewood Cliffs, NJ, 1977.