Tools & Notions for Program Construction

An advanced course

J.J. Arsac
J.P.G. Barnes
D. Bert
P.J. Brown
P. Cousot
R. Cousot
Ph. Deschamp
M.J.C. Gordon
A.N. Habermann
M.A. Jackson
M. Lemaitre
M. Lemoine
B. Lorho
J.C. Reynolds
B. Sufrin
D.A. Turner
G. Zanon

Edited by D. Néel

Tools and notions for program construction

An advanced course

Edited by

D. NÉEL

Institute National de Recherche en Informatique et en Automatique

Cambridge University Press
Cambridge
London New York New Rochelle
Melbourne Sydney

INDUCTION PRINCIPLES FOR PROVING INVARIANCE PROPERTIES OF PROGRAMS

Patrick COUSOT* & Radhia COUSOT**

ABSTRACT : We propose sixteen sound and complete induction principles for proving program invariance properties. We study their relationships and show that they can be derived from each others by commuting mathematical transformations. Only five of these induction principles correspond to already known invariance proof methods. We choose a non-conventional induction principle and construct corresponding partial correctness, non-termination and clean behavior proof methods. When constructing these new proof methods, we informally apply Cousot & Cousot[80b] mathematical approach. This essentially consists in decomposing the global inductive invariant involved in the induction principle into an equivalent set of local invariants and in deriving the corresponding verification conditions.

1. INTRODUCTION

In an introductory series of examples, we explore a number of partial correctness proof methods including the so-called intermediate assertion method of Floyd[67]-Naur[66], the subgoal induction method of Morris & Wegbreit[77], their contrapositive versions and a number of variants (Ch.2).

When informally explained using examples, the essence of these proof methods is difficult to capture. Therefore we present an abstract model for the study, comparison and systematic contruction of program proof methods.

We illustate our approach using a sequential iterative language which had to be chosen simple enough in order to avoid lengthy formal computations. However our approach is independent from any particular programming language and has been, for example, applied to parallel programs (Cousot & Cousot[80a], [80b], Cousot R.[81]). This is because we use discrete state transition systems (Cousot P.[79]) as abstract models of programs (Ch.3).

---

* Université de Metz, Faculté des Sciences, Ile du Saulcy, 57000 Metz, France.
** Attaché de Recherche au CNRS, CRIN-Nancy, France.

We consider in this paper, an important class of program properties called invariance properties. This class of properties includes partial correctness, non-termination, clean-behavior (absence of run-time errors), etc. (Ch.4).

We show how the essence of a proof method can be captured as a basic induction principle (Ch.5).
We propose sixteen sound and complete induction principles for proving invariance properties of programs which we classify as relational or assertional, predictive or deductive, forward or backward and positive or contrapositive. Only five of these induction principles correspond to already known proof methods. We show that these induction principles can be derived from each other using mathematical transformations. Other kind of relationships between the induction principles (and hence the corresponding proof methods) include equivalence in the sense that a proof by either of these methods can be used to produce a proof by any other method (Ch.6).

In Cousot & Cousot[80b] we have studied a mathematical method for constructing proof methods. This essentially consists in defining the operational semantics of the programming language, choosing an induction principle for the considered class of properties, choosing a decomposition of the global inductive invariant involved in the induction principle into an equivalent set of local invariants and finally deriving the verification conditions corresponding to that decomposition from the induction principle. This mathematical method for constructing program proof methods is informally reviewed in Ch.7.

We next define the semantics of our example programming language (Ch.8), choose a non-conventional induction principle and construct sound and complete methods for proving non-termination (Ch.9), clean-behavior (Ch.10) and global invariants (Ch.11) of programs.

We conclude (Ch.12) that complementary methods should be used for proving all usual different invariance properties of a program. Here complementary means that a single set of invariant assertions can be used for each of the properties which are considered for the program. We show that this is the case of the induction principle underlying Floyd's method but not Morris & Wegbreit's subgoal induction methods. In our opinion this explains the relative success of Floyd's partial correctness proof method.

2. PROVING THE PARTIAL CORRECTNESS OF AN INTRODUCTORY EXAMPLE BY SEVERAL DIFFERENT METHODS

We illustrate a series of different partial correctness proof methods using the following program fragment which computes the quotient q and remainder r of two integers x and y (so that x=q*y+r and r<y). This example has been chosen naïve and well-known enough so that it is simple to understand and its partial correctness is obvious. Therefore the reader can easily concentrate on studying the various partial correctness proof methods which will be illustrated.

```
1: Q:=0;
2: R:=X;
3: while R≥Y do
4:     Q:=Q+1;
5:     R:=R-Y;
6: od;
7:
```

(Variables are implicitly declared as integer variables, without bounds on their possible values. The "input" variables X and Y are assumed to be initialized. Labels have been used so that program points can easily be designated).

Proving that this program is partially correct consists in showing that if execution starts at program point 1 with initial values $\underline{x},\underline{y}$ for the variables X,Y and terminates with final values $\overline{x},\overline{y},\overline{q},\overline{r}$ of the variables X,Y,Q,R then at termination point 7 we have $(\overline{x}=\underline{x} \wedge \overline{y}=\underline{y} \wedge \underline{x}=\overline{q}*\underline{y}+\overline{r} \wedge \overline{r}<\underline{y})$.

2.1 ASSERTIONAL FORWARD DEDUCTIVE POSITIVE METHOD ($\mathcal{A}$)

In Floyd[67]-Naur[66]'s method one associates an intermediate assertion P_i with each program point i,i=1,...,7. The meaning of an assertion $P_i(x,y,q,r)$ is that it describes (a superset of) the possible values x,y,q,r of the variables X,Y,Q,R whenever control reaches program point i :

$P_1(x,y,q,r) = \underline{true}$
$P_2(x,y,q,r) = [q=0]$
$P_3(x,y,q,r) = [q=0 \wedge r=x]$
$P_4(x,y,q,r) = [x=q*y+r]$
$P_5(x,y,q,r) = [x=q*y+(r-y)]$
$P_6(x,y,q,r) = [x=q*y+r]$
$P_7(x,y,q,r) = [x=q*y+r \wedge r<y]$

These intermediate assertions must be shown to satisfy the following verification conditions :

(i_ε) $P_1(x,y,q,r) \Leftarrow \underline{\text{true}}$

(i_2) $P_2(x,y,q,r) \Leftarrow [\exists q' | P_1(x,y,q',r) \wedge q=0]$

(i_3) $P_3(x,y,q,r) \Leftarrow [\exists r' | P_2(x,y,q,r') \wedge r=x]$

(i_4) $P_4(x,y,q,r) \Leftarrow [(P_3(x,y,q,r) \vee P_6(x,y,q,r)) \wedge (r \geq y)]$

(i_5) $P_5(x,y,q,r) \Leftarrow [\exists q' | P_4(x,y,q',r) \wedge q=q'+1]$

(i_6) $P_6(x,y,q,r) \Leftarrow [\exists r' | P_5(x,y,q,r') \wedge r=r'-y]$

(i_7) $P_7(x,y,q,r) \Leftarrow [(P_3(x,y,q,r) \vee P_6(x,y,q,r)) \wedge (r<y)]$

(i_σ) $[x=q*y+r \wedge r<y] \Leftarrow P_7(x,y,q,r)$

As a simple exercise, we leave to the care of the reader the proof that for all possible values of x,y,q,r, the intermediate assertions satisfy these verification conditions. Then according to Floyd-Naur's method the partial correctness proof is finished.

In order to understand why, notice that the verification conditions have been defined so that :

- $P_1(x,y,q,r)$ must be true when execution begins.
- Whenever execution reaches some program point i which is immediately followed by program point j, then the assumption that $P_i(x,y,q,r)$ is true when control is at point i implies that $P_j(x,y,q,r)$ must then be true when control reaches program point j.

For example, if execution reaches program point 3 with $P_3(x,y,q,r)$ assumed to hold, then either $r \geq y$ and control will reach program point 4 with $P_4(x,y,q,r)$ holding (because of verification condition i_4) or $r<y$ and control will reach point 7 with $P_7(x,y,q,r)$ holding (because of verification condition i_7).

The same way, if execution reaches program point 4 with $P_4(x',y',q',r')$ assumed to hold where x',y',q',r' are the values of the variables X,Y,Q,R at this point. Then control goes to program point 5 with new values x,y,q,r of the variables such that $x=x' \wedge y=y' \wedge q=q'+1 \wedge r=r' \wedge P_4(x',y',q',r')$ that is $P_4(x,y,q',r) \wedge q=q'+1$. Then because of verification condition i_5 we can conclude that $P_5(x,y,q,r)$ is true.

Now the program has been shown to be partially correct because the verification conditions allow us to conclude that whenever control reaches some program point i then $P_i(x,y,q,r)$ must be true (therefore in particular upon termination at program point 7). The proof is by induction on the number n of computation steps since the computation started at program point 1. For the basis, that is after n=0 computation steps, control is at program point 1 and $P_1(x,y,q,r)$ holds by verification condition i_ε. For the induction, after n computation steps either program execution is terminated and we have done or control is at some program point i, $i \neq 7$. By induction hypothesis we can assume that P_i holds. The n+1-th computation leads to some immediately following

program point j. Then P_j must hold when control reaches program point j because P_i is true by induction hypothesis and P_j satisfies verification condition i_j.

2.2 ASSERTIONAL FORWARD PREDICTIVE POSITIVE METHOD ($\tilde{\iota}$)

Hoare[69]'s partial correctness proof method is similar to Floyd-Naur's method in that the same intermediate assertions are used. Also, the proof that Hoare's method is sound is similar to the one of Floyd's method, since one must prove that whenever control reaches some program point i then $P_i(x,y,q,r)$ is true. However, the verification conditions are somewhat different :

$P_1(x,y,q,r) \Leftarrow \underline{\text{true}}$

$P_1(x,y,q,r) \Rightarrow P_2(x,y,0,r)$
$P_2(x,y,q,r) \Rightarrow P_3(x,y,q,x)$
$P_3(x,y,q,r) \Rightarrow [(P_4(x,y,q,r) \wedge r \geq y) \vee (P_7(x,y,q,r) \wedge r<y)]$
$P_4(x,y,q,r) \Rightarrow P_5(x,y,q+1,r)$
$P_5(x,y,q,r) \Rightarrow P_6(x,y,q,r-y)$
$P_6(x,y,q,r) \Rightarrow [(P_4(x,y,q,r) \wedge r \geq y) \vee (P_7(x,y,q,r) \wedge r<y)]$

$[x=q*y+r \wedge r<y] \Leftarrow P_7(x,y,q,r)$

The above presentation of Hoare's method is somewhat unfaithfull. The main reason is that we have not defined the verification conditions by induction on the syntax of programs. Another reason is that we used a predictive form for all verification conditions as opposed to Hoare who introduced a predictive verification condition for assignment but used Floyd's deductive verification conditions for conditional and while commands.

Both deductive and predictive forward verification conditions guarantee that whenever execution reaches some program point i (with values x,y,q,r for variables X,Y,Q,R) which is immediately followed by program point j (where the values x',y',q',r' of X,Y,Q,R are related to the values of the variables at point j by say $t_{ij}(x,y,q,r,x',y',q',r')$) then the assumption that $P_i(x,y,q,r)$ is true when control is at point i implies that $P_j(x',y',q',r')$ must be true when control reaches program point j.

In the deductive verification condition, one assumes that $P_i(x,y,q,r)$ holds and deduces the strongest consequence at point j and this consequence (i.e. $P_i(x,y,q,r) \wedge t_{ij}(x,y,q,r,x',y',q',r')$) must imply $P_j(x',y',q',r')$.

In the predictive verification condition, one predicts at program point i what is the weakest condition for $P_j(x',y',q',r')$ to hold when control goes from i to j, and the assumption that $P_i(x,y,q,r)$ holds must imply this predicted condition (i.e. $t_{ij}(x,y,q,r,x',y',q',r') \wedge P_j(x',y',q',r')$).

2.3 RELATIONAL FORWARD DEDUCTIVE POSITIVE METHOD (I)

The partial correctness of the following program cannot be expressed as an assertion upon the final values of the variables (because the initial value of X is not memorized) :

```
1: Q:=0;
2: while X≥Y do
3:     Q:=Q+1;
4:     X:=X-Y;
5: od;
6:
```

In order to use Floyd-Naur's partial correctness proof method, one must introduce an auxiliary variable XI which is assigned the value of X on program entry and never modified afterwards :

XI:=X; Q:=0; while X≥Y do Q:=Q+1; X:=X-Y; od

so that upon termination we can prove that $[xi=q*y+x \wedge x<y]$.

Such a trick can be avoided by using Manna[71]'s partial correctness proof method. The method is similar to Floyd's method except that the intermediate assertion P_i associated with program point i expresses a relationship between the initial values $\underline{x},\underline{y}$ of variables X and Y and the current values x,y,q of these variables X,Y,Q which is true whenever control reaches program point i :

$P_1(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y}]$
$P_2(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y} \wedge q=0]$
$P_3(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=q*\underline{y}+x]$
$P_4(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=(q-1)*\underline{y}+x]$
$P_5(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=q*\underline{y}+x]$
$P_6(\underline{x},\underline{y},x,y,q) = [\underline{x}=q*\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}]$

The verification conditions are similar to the ones of Floyd-Naur's method except for the entry intermediate assertion :

$P_1(\underline{x},\underline{y},x,y,q) \Leftarrow [x=\underline{x} \wedge y=\underline{y}]$

$P_2(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q' | P_1(\underline{x},\underline{y},x,y,q') \wedge q=0]$
$P_3(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge x \geq y]$
$P_4(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q' | P_3(\underline{x},\underline{y},x,y,q') \wedge q=q'+1]$
$P_5(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists x' | P_4(\underline{x},\underline{y},x',y,q) \wedge x=x'-y]$
$P_6(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge x<y]$

$[\underline{x}=q*\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}] \Leftarrow P_6(\underline{x},\underline{y},x,y,q)$

2.4 RELATIONAL FORWARD PREDICTIVE POSITIVE METHOD ($\tilde{I}$)

The above deductive verification conditions are equivalent to the following predictive ones :

$[x=\underline{x} \wedge y=\underline{y}] \Rightarrow P_1(\underline{x},\underline{y},x,y,q)$

$P_1(\underline{x},\underline{y},x,y,q) \Rightarrow P_2(\underline{x},\underline{y},x,y,0)$
$P_2(\underline{x},\underline{y},x,y,q) \Rightarrow [(P_3(\underline{x},\underline{y},x,y,q) \wedge x \geq y) \vee (P_6(\underline{x},\underline{y},x,y,q) \wedge x<y)]$
$P_3(\underline{x},\underline{y},x,y,q) \Rightarrow P_4(\underline{x},\underline{y},x,y,q+1)$
$P_4(\underline{x},\underline{y},x,y,q) \Rightarrow P_5(\underline{x},\underline{y},x-y,y,q)$
$P_5(\underline{x},\underline{y},x,y,q) \Rightarrow [(P_3(\underline{x},\underline{y},x,y,q) \wedge x \geq y) \vee (P_6(\underline{x},\underline{y},x,y,q) \wedge x<y)]$

$P_6(\underline{x},\underline{y},x,y,q) \Rightarrow [\underline{x}=q*\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}]$

2.5 RELATIONAL BACKWARD DEDUCTIVE POSITIVE METHOD (I^{-1})

In Morris & Wegbreit[77]'s partial correctness method so-called subgoal induction, the assertion P_i associated with program point i relates the values x,y,q of the variables X,Y,Q with the values $\overline{x},\overline{q}$ of X,Q on termination. Therefore instead of describing what has been done by the program from the entry point up to program point i, the intermediate assertion P_i describes what remains to be done from program point i up to the end of execution :

$P_1(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) = [x=\underline{\overline{q}}*y+\underline{\overline{x}} \wedge \underline{\overline{x}}<y]$
$P_2(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) = [x=(\underline{\overline{q}}-q)*y+\underline{\overline{x}} \wedge \underline{\overline{x}}<y]$
$P_3(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) = [x=(\underline{\overline{q}}-q)*y+x \wedge \underline{\overline{x}}<\underline{y}]$
$P_4(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) = [x=(\underline{\overline{q}}-q+1)*\underline{y}+\underline{\overline{x}} \wedge \underline{\overline{x}}<y]$
$P_5(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) = [x=(\underline{\overline{q}}-q)*\underline{y}+x \wedge \underline{\overline{x}}<y]$
$P_6(x,y,q,\overline{x},\overline{q}) = [x=\overline{x} \wedge q=\overline{q}]$

The meaning of intermediate assertion $P_i(x,y,q,\overline{x},\overline{y})$ is that if control reaches program point i with values x,y,q for the variables X,Y,Q and if afterwards the execution of the program terminates with final values $\overline{x},\overline{q}$ for X and Q, then $P_i(x,y,q,\overline{x},\overline{q})$ must be true.

The deductive verification conditions which ensure this property are the following :

(I_ε^{-1}) $[x=\underline{\overline{q}}*y+\underline{\overline{x}} \wedge \underline{\overline{x}}<y] \Leftarrow P_1(x,y,q,\underline{\overline{x}},\underline{\overline{q}})$

(I_1^{-1}) $P_1(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \Leftarrow P_2(x,y,0,\underline{\overline{x}},\underline{\overline{q}})$
(I_2^{-1}) $P_2(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \Leftarrow [(P_3(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \wedge x \geq y) \vee (P_6(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \wedge x<y)]$
(I_3^{-1}) $P_3(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \Leftarrow P_4(x,y,q+1,\underline{\overline{x}},\underline{\overline{q}})$
(I_4^{-1}) $P_4(x,y,q,\underline{\overline{x}},\underline{\overline{q}}) \Leftarrow P_5(x-y,y,q,\underline{\overline{x}},\underline{\overline{q}})$
(I_5^{-1}) $P_5(x,y,q,\overline{x},\overline{q}) \Leftarrow [(P_3(x,y,q,\overline{x},\overline{q}) \wedge x \geq y) \vee (P_6(x,y,q,\overline{x},\overline{q}) \wedge x<y)]$

(I_σ^{-1}) $P_6(x,y,q,\overline{x},\overline{q}) \Leftarrow [x=\overline{x} \wedge q=\overline{q}]$

When the intermediate assertions have been shown to satisfy these verification conditions, the partial correctness proof is finished.

In order to understand why, notice that the verification conditions have been defined so that :

- $P_6(x,y,q,\overline{x},\overline{q})$ must be true when execution ends.
- Whenever execution reaches some program point j which was immediately preceded by program point i, then the assumption that $P_j(x,y,q,\overline{x},\overline{q})$ holds (between the values x,y,q of X,Y,Q at point j and $\overline{x},\overline{q}$ of X,Q upon termination) must imply that $P_i(x,y,q,\overline{x},\overline{q})$ was holding (between the values x,y,q of X,Y,Q at point i and $\overline{x},\overline{q}$ of X,Q upon termination) when control reached program point i.

For example if execution reaches program point 3 with $P_3(x,y,q,\overline{x},\overline{q})$ assumed to hold, then execution came either from program point 2 and $P_2(x,y,q,\overline{x},\overline{q})$ was true (because of verification condition I_2^{-1}) or from program point 5 and $P_5(x,y,q,\overline{x},\overline{q})$ was true (because of verification condition I_5^{-1}).

The same way if execution reaches program point 4 with x',y',q' as values of the variables X,Y,Q at that program point 4. Then execution came from program point 3 where the values x,y,q of the variables X,Y,Q were such that x'=x, y'=y, q'=q+1. If $P_4(x',y',q',\overline{x},\overline{q})$ is assumed to hold at program point 4, then $P_4(x,y,q+1,\overline{x},\overline{q})$ holds at program point 3. Then because of verification condition I_3^{-1} we can conclude that $P_3(x,y,q,\overline{x},\overline{q})$ was holding when control was at program point 3.

Now the program has been shown to be partially correct because the verification conditions allow us to conclude that whenever control reaches some program point i with X=x, Y=y and Q=q and afterwards execution terminates with $X=\overline{x}$ and $Q=\overline{y}$ then $P_i(x,y,q,\overline{x},\overline{q})$ was true (therefore in particular, when execution began at program point 1). The proof is by induction on the number n of computation steps until the computation halts at program point 6. For the basis, that is when n=0 computation steps remain before termination, control is at program point 6 and $P_6(x,y,q,\overline{x},\overline{q})$ holds by verification condition I_6^{-1}. For the induction, when n computation steps remain either program execution is starting and we have done or control is at some program point j, $j\neq 1$. By induction hypothesis, we can assume that P_j holds. When n+1 computation steps were remaining, control was at some program point i immediately preceding point j. Then P_i had to hold when control reached program point i because P_j is true by induction hypothesis and P_i satisfies verification condition I_i^{-1}.

2.6 RELATIONAL BACKWARD PREDICTIVE POSITIVE METHOD ($I^{\sim -1}$)

The deductive verification conditions of Morris & Wegbreit's subgoal induction method are equivalent to the following predictive ones :

$$[x=\overline{q}*y+\overline{x} \wedge \overline{x}<y] \Leftarrow P_1(x,y,q,\overline{x},\overline{q})$$

$$P_2(x,y,q,\underline{x},\underline{q}) \Rightarrow [\exists q' | P_1(x,\underline{y},\underline{q}',\underline{x},\underline{q}) \wedge q=0]$$

$$P_3(x,y,q,\underline{x},\underline{q}) \Rightarrow [(P_2(x,y,q,x,q) \vee P_5(x,y,q,\underline{x},\underline{q})) \wedge (x \leq y)]$$

$$P_4(x,y,q,\underline{x},\underline{q}) \Rightarrow [\exists q' | P_3(x,y,q',\underline{x},\underline{q}) \wedge q=q'+1]$$

$$P_5(x,y,q,\underline{x},\underline{q}) \Rightarrow [\exists x' | P_4(x',\underline{y},\underline{q},x,q) \wedge x=x'-\underline{y}]$$

$$P_6(x,y,q,\underline{x},q) \Rightarrow [(P_2(x,y,q,x,q) \vee P_5(x,y,q,\underline{x},\underline{q})) \wedge (x>y)]$$

$$P_6(x,y,q,\overline{x},\overline{q}) \Leftarrow [x=\overline{x} \wedge y=\overline{y}]$$

Both deductive and predictive forward verification conditions guarantee that whenever execution reaches some program point j (with x',y',q' as values for variables X,Y,Q) which was immediately preceded by program point i (with x,y,q as values for variables X,Y,Q such that say $t_{ij}(x,y,q,x',y',q')$) then the assumption that $P_j(x',y',q',\overline{x},\overline{q})$ holds at point j must imply that $P_i(x,y,q,\overline{x},\overline{q})$ was holding when control reached program point i.

In the deductive verification condition, one assumes that $P_j(x',y',q',\overline{x},\overline{q})$ holds and deduces the strongest consequence at point i and this consequence (i.e. $t_{ij}(x,y,q,x',y',q') \wedge P_j(x',y',q',\overline{x},\overline{q})$) must imply $P_i(x,y,q,\overline{x},\overline{q})$.

In the predictive verification condition, one predicts at program point j what is the weakest condition for $P_i(x',y',q',\overline{x},\overline{q})$ to hold when control goes from i to j, and the assumption that $P_j(x',y',q',\overline{x},\overline{q})$ holds must imply this predicted condition (i.e. $P_i(x,y,q,\overline{x},\overline{q}) \wedge t_{ij}(x,y,q,x',y',q')$).

2.7 ASSERTIONAL BACKWARD DEDUCTIVE POSITIVE METHOD ($\mathcal{A}^{-1}$)

This partial correctness proof method is a special case of Morris & Wegbreit's subgoal induction where final values of variables are not used. Then relationships between initial and final values of variables cannot be directly expressed. One can only prove that if the program terminates then the initial values of the variables had to satisfy some condition (notice that this condition is necessary for termination but not sufficient).

For example, we can prove that if the program

```
1:
2:  Q:=0;
3:  while X≥Y do
4:      Q:=Q+1;
5:      X:=X-Y;
6:  od;
```

terminates then necessarily $(x<y) \vee (x \geq y \wedge y \geq 0)$

The intermediate assertions are :

$P_1(x,y,q) = [(x \geq y) \Rightarrow (y \geq 0)]$
$P_2(x,y,q) = [(x \geq y) \Rightarrow (y \geq 0)]$
$P_3(x,y,q) = [(x \geq 2*y) \Rightarrow (y \geq 0)]$
$P_4(x,y,q) = [(x \geq 2*y) \Rightarrow (y \geq 0)]$
$P_5(x,y,q) = [(x \geq y) \Rightarrow (y \geq 0)]$
$P_6(x,y,q) = \underline{true}$

and the verification conditions are :

$[(x<y) \vee (x \geq y \wedge y \geq 0)] \Leftarrow P_1(x,y,q)$

$P_1(x,y,q) \Leftarrow P_2(x,y,0)$
$P_2(x,y,q) \Leftarrow [(P_3(x,y,q) \wedge x \geq y) \vee (P_6(x,y,q) \wedge x<y)]$
$P_3(x,y,q) \Leftarrow P_4(x,y,q+1)$
$P_4(x,y,q) \Leftarrow P_5(x-y,y,q)$
$P_5(x,y,q) \Leftarrow [(P_3(x,y,q) \wedge x \geq y) \vee (P_6(x,y,q) \wedge x<y)]$

$P_6(x,y,q) \Leftarrow \underline{true}$

When the definition of partial correctness of a program involves a relationship between initial and final values of the variables, we have mentionned that Floyd's method can be used on an equivalent program which is the original one transformed by an assignment on entry of the initial values of variables to auxiliary variables. A similar idea for the assertional backward positive method would consist in memorizing the final values of X and Q by an assignment to auxiliary variables XF and QF on program exit. This does not work. In fact the backward symmetric of the forward trick consists in terminating the program by **while** (XF≠X **or** QF≠Q) **do** **skip**;**od**. This transformed program is equivalent to the original one with respect to partial correctness. The intermediate assertions would be :

{xf=x ∧ qf=q} **while** (XF≠X **or** QF≠Q) **do** {xf=x ∧ qf=q} **skip**; {xf=x ∧ qf=q} **od** {**true**}.
It is clear that the relational proof method should be prefered to such a trick!

2.8 ASSERTIONAL BACKWARD PREDICTIVE POSITIVE METHOD ($\tilde{\iota}^{-1}$)

The predictive version of the above deductive verification conditions is left to the reader.

2.9 RELATIONAL FORWARD DEDUCTIVE CONTRAPOSITIVE METHOD ($\overline{I^{-1}}$)

This partial correctness proof method is contrapositive, in that the following intermediate assertions describe what will not happen during program execution :

$P_1(x,y,q,\overline{x},\overline{q}) = [(x=\overline{q}*y+\overline{x}) \Rightarrow (\overline{x} \geq y)]$
$P_2(x,y,q,\overline{x},\overline{q}) = [(x=(\overline{q}-q)*y+\overline{x}) \Rightarrow (\overline{x} \geq y)]$
$P_3(x,y,q,\overline{x},\overline{q}) = [(x=(\overline{q}-q)*y+\overline{x}) \Rightarrow (\overline{x} \geq \underline{y})]$
$P_4(x,y,q,\overline{x},\overline{q}) = [(x=(\overline{q}-q+1)*\underline{y}+\overline{x}) \Rightarrow (\overline{x} \geq y)]$
$P_5(x,y,q,\overline{x},\overline{q}) = [(x=(\overline{q}-q)*y+\overline{x}) \Rightarrow (\overline{x} \geq y)]$
$P_6(x,y,q,\overline{x},\overline{q}) = [((x=(\overline{q}-q)*y+\overline{x}) \Rightarrow (\overline{x} \geq y)) \wedge (x<y)]$

Let x,y,q and $\overline{x},\overline{y},\overline{q}$ be the initial and final values of the program variables X,Y,Q. If the program were not partially correct then $\neg[x=\overline{q}*y+\overline{x} \wedge \overline{x}<y]$ would hold and $P_1(x,y,q,\overline{x},\overline{q})$ would be true. This follows from the first verification condition :

$$P_1(x,y,q,\overline{x},\overline{q}) \Leftarrow \neg[x=\overline{q}*y+\overline{x} \wedge \overline{x}<y]$$

Then by induction on the number n of computation steps during program execution, the hypothesis that $P_1(x,y,q,\overline{x},\overline{q})$ holds and the following verification conditions would imply that $P_i(x,y,q,\overline{x},\overline{q})$ holds for i=1,...,6 :

$P_2(x,y,q,\overline{x},\overline{q}) \Leftarrow [\exists q' | P_1(x,\underline{y},\underline{q}',\overline{x},\overline{q}) \wedge q=0]$
$P_3(x,y,q,\overline{x},\overline{q}) \Leftarrow [(P_2(x,y,q,\overline{x},\overline{q}) \vee \underline{P_5}(x,y,q,\overline{x},\overline{q})) \wedge (x \leq y)]$
$P_4(x,y,q,\overline{x},\overline{q}) \Leftarrow [\exists q' | P_3(x,y,q',\underline{x},\overline{q}) \wedge q=q'+1]$
$P_5(x,y,q,\overline{x},\overline{q}) \Leftarrow [\exists x' | P_4(x',\underline{y},\underline{q},\overline{x},\overline{q}) \wedge x=x'-\underline{y}]$
$P_6(x,y,q,\overline{x},\overline{q}) \Leftarrow [(P_2(x,y,q,\overline{x},\overline{q}) \vee P_5(x,y,q,\overline{x},\overline{y})) \wedge (x>y)]$

If moreover, we assume that execution of the program terminates, then the last verification condition :

$$\neg P_6(x,y,q,\overline{x},\overline{q}) \Leftarrow [x=\overline{x} \wedge q=\overline{q}]$$

would ensure that $P_6(x,y,q,\overline{x},\overline{q})$ is not true. Since termination would lead to a contradiction we have proved by reductio ad absurdum that the program is partially correct.

2.10 RELATIONAL FORWARD PREDICTIVE CONTRAPOSITIVE METHOD ($\widetilde{\overline{I^{-1}}}$)

The reader is now familiar with the fact that the above deductive verification conditions are equivalent to the following predictive ones :

$$\neg[x=\overline{q}*y+\overline{x} \wedge \overline{x}<y] \Rightarrow P_1(x,y,q,\overline{x},\overline{q})$$

$$P_1(x,y,q,\underline{x},\underline{q}) \Rightarrow P_2(x,y,0,\overline{x},\overline{q})$$
$$P_2(x,y,q,\underline{x},\underline{q}) \Rightarrow [(P_3(x,y,q,\underline{x},\underline{q}) \wedge x\geq y) \vee (P_6(x,y,q,\overline{x},\overline{q}) \wedge x<y)]$$
$$P_3(x,y,q,\underline{x},\underline{q}) \Rightarrow P_4(x,y,q+1,\underline{x},\underline{q})$$
$$P_4(x,y,q,\underline{x},\underline{q}) \Rightarrow P_5(x-y,y,q,\underline{x},\underline{q})$$
$$P_5(x,y,q,x,q) \Rightarrow [(P_3(x,y,q,\underline{x},\underline{q}) \wedge x\geq y) \vee (P_6(x,y,q,\overline{x},\overline{q}) \wedge x<y)]$$
$$[x=\overline{x} \wedge q=\overline{q}] \Rightarrow \neg P_6(x,y,q,\overline{x},\overline{q})$$

2.11 RELATIONAL BACKWARD DEDUCTIVE CONTRAPOSITIVE METHOD ($\overline{I}$)

This partial correctness proof method is also contrapositive, in that the following intermediate assertions describe what has not happened during program execution :

$$P_1(\underline{x},\underline{y},x,y,q) = [(x=\underline{x}) \Rightarrow (y\neq\underline{y})]$$
$$P_2(\underline{x},\underline{y},x,y,q) = [(x=\underline{x} \wedge q=0) \Rightarrow (y\neq\underline{y})]$$
$$P_3(\underline{x},\underline{y},x,y,q) = [(\underline{x}=q*\underline{y}+x) \Rightarrow (y\neq\underline{y})]$$
$$P_4(\underline{x},\underline{y},x,y,q) = [(\underline{x}=(q-1)*\underline{y}+x) \Rightarrow (y\neq\underline{y})]$$
$$P_5(\underline{x},\underline{y},x,y,q) = [(\underline{x}=q*\underline{y}+x) \Rightarrow (y\neq\underline{y})]$$
$$P_6(\underline{x},\underline{y},x,y,q) = [(\underline{x}=q*\underline{y}+x \wedge x<y) \Rightarrow (y\neq\underline{y})]$$

Let $\underline{x},\underline{y},\underline{q}$ and x,y,q be the initial and final values of the program variables X,Y,Q. If the program were not partially correct then $\neg[\underline{x}=q*\underline{y}+x \wedge x<y \wedge y=\underline{y}]$ would hold and $P_6(\underline{x},\underline{y},x,y,q)$ would be true. This follows from the last verification condition :

$$P_6(\underline{x},\underline{y},x,y,q) \Leftarrow \neg[\underline{x}=q*\underline{y}+x \wedge x<y \wedge y=\underline{y}]$$

Then, by induction on the number of computation steps until the computation halts at program point 6, the hypothesis that $P_6(\underline{x},\underline{y},x,y,q)$ holds and the following verification conditions would imply that if program execution started with initial values $\underline{x},\underline{y}$ of X,Y next reaches some program point i with x,y,q as current values of X,Y,Q and subsequently terminates then $P_i(\underline{x},\underline{y},x,y,q)$ would hold :

$$P_1(\underline{x},\underline{y},x,y,q) \Leftarrow P_2(\underline{x},\underline{y},x,y,0)$$
$$P_2(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_3(\underline{x},\underline{y},x,y,q) \wedge x\geq y) \vee (P_6(\underline{x},\underline{y},x,y,q) \wedge x<y)]$$
$$P_3(\underline{x},\underline{y},x,y,q) \Leftarrow P_4(\underline{x},\underline{y},x,y,q+1)$$
$$P_4(\underline{x},\underline{y},x,y,q) \Leftarrow P_5(\underline{x},\underline{y},x-y,y,q)$$
$$P_5(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_3(\underline{x},\underline{y},x,y,q) \wedge x\geq y) \vee (P_6(\underline{x},\underline{y},x,y,q) \wedge x<y)]$$

If moreover we assume that execution of the program terminates then the first verification condition :

$$\neg P_1(\underline{x},\underline{y},x,y,q) \Leftarrow [x=\underline{x} \wedge y=\underline{y}]$$

would ensure that $P_1(\underline{x},\underline{y},x,y,q)$ is not true. Since termination would lead to a contradiction we have proved by reductio ad absurdum that the program is partially correct.

2.12 RELATIONAL BACKWARD PREDICTIVE CONTRAPOSITIVE METHOD $(\tilde{\tilde{I}})$

The predictive version of the above deductive verification conditions is :

$$\neg P_1(\underline{x},\underline{y},x,y,q) \Leftarrow [x=\underline{x} \wedge y=\underline{y}]$$

$$P_2(\underline{x},\underline{y},x,y,q) \Rightarrow [\exists q' | P_1(\underline{x},\underline{y},x,y,q') \wedge q=0]$$
$$P_3(\underline{x},\underline{y},x,y,q) \Rightarrow [(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge (x \geq y)]$$
$$P_4(\underline{x},\underline{y},x,y,q) \Rightarrow [\exists q' | P_3(\underline{x},\underline{y},x,y,q') \wedge q=q'+1]$$
$$P_5(\underline{x},\underline{y},x,y,q) \Rightarrow [\exists x' | P_4(\underline{x},\underline{y},x',y,q) \wedge x=x'-y]$$
$$P_6(\underline{x},\underline{y},x,y,q) \Rightarrow [(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge (x<y)]$$

$$P_6(\underline{x},\underline{y},x,y,q) \Leftarrow \neg[\underline{x}=q*\underline{y}+x \wedge x<\underline{y} \wedge \underline{y}=y]$$

2.13 ASSERTIONAL CONTRAPOSITIVE METHODS

The reader can easily conceive the assertional counterparts of the relational contrapositive methods.
Contrapositive methods are not well-suited for proving positive properties of programs but sometimes turn out to be quite natural for proving negative properties such as non-termination.

3. PROGRAMS AS TRANSITION RELATIONS ON STATES

An essential step in understanding invariance proof methods consists in considering a model of programs where all properties of programs which are not relevant to the proof methods can be left out. For this purpose we will forget everything about programs except that a program P defines a set $S[\![P]\!]$ of states and a transition relation $t[\![P]\!] \in (S[\![P]\!] \times S[\![P]\!]) \rightarrow \{tt,ff\})$. Moreover the initial states will be specified by a characteristic predicate $\varepsilon[\![P]\!] \in (S[\![P]\!] \rightarrow \{tt,ff\})$, whereas the final states are characterized by $\sigma[\![P]\!] \in (S[\![P]\!] \rightarrow \{tt,ff\})$.

Let us consider our example program :

```
1: Q:=0;
2: while X≥Y do
3:     Q:=Q+1;
4:     X:=X-Y;
5: od;
6:
```

The states <c,m> of this program consist of a control state c (i.e. a program point) and a memory state (i.e. a function m which defines the value m(X),m(Y),m(Q) of the variables X,Y,Q) :

$C = \{1,2,3,4,5,6\}$
$M = \{X,Y,Q\} \to \mathbb{Z}$ where $\mathbb{Z}$ is the set of integers
$S = C \times M$

The initial states $\underline{s}$ of the program correspond to program point 1 with arbitrary values for the variables :

$$\varepsilon(\underline{s}) = [\exists m \in M | \underline{s} = <1,m>]$$

and the final states $\overline{s}$ to program point 6 :

$$\sigma(\overline{s}) = [\exists m \in M | \overline{s} = <6,m>]$$

The transition relation $t \in (S \times S) \to \{tt,ff\}$ is true between a state and its possible successor during program execution. More precisely $t(<i,m>,<j,m'>)$ is true if and only if whenever, during program execution, control is at program point i in memory state m then after the next computation step, control can be at program point j in memory state m'.

Let us now define the transition relation corresponding to the above program. We will write $<i,m> \overset{t}{\to} <j,m'>$ whenever $t(<i,m>,<j,m'>)$ is true, distinguishing between several cases according to the value of i. Moreover (x,y,q) stands for the function m such that m(X)=x, m(Y)=y and m(Q)=q and these values of program variables X,Y,Q are implicitly universally quantified over integers :

$<1,(x,y,q)> \overset{t}{\to} <2,(x,y,0)>$
$<2,(x,y,q)> \overset{t}{\to} <3,(x,y,q)>$ iff $x \geq y$
$<2,(x,y,q)> \overset{t}{\to} <6,(x,y,q)>$ iff $x < y$
$<3,(x,y,q)> \overset{t}{\to} <4,(x,y,q+1)>$
$<4,(x,y,q)> \overset{t}{\to} <5,(x-y,y,q)>$
$<5,(x,y,q)> \overset{t}{\to} <3,(x,y,q)>$ iff $x \geq y$
$<5,(x,y,q)> \overset{t}{\to} <6,(x,y,q)>$ iff $x < y$

The reflexive transitive closure t^* of t is defined as follows :

$t^0(s,s') = (s=s')$
$t^{n+1}(s,s') = [\exists s''\in S | t^n(s,s'') \wedge t(s'',s')]$ when $n\geq 0$
$t^*(s,s') = [\exists n\geq 0 | t^n(s,s')]$

so that $t^*(s,s')$ is true iff s' is a program state which can be reached from state s after $n\geq 0$ computation steps as defined by t.

Let us define :

$\overline{\psi}(\underline{x},\underline{y},\underline{q},\overline{x},\overline{y},\overline{q}) = [\underline{x}=\overline{q}*\underline{y}+\overline{x} \wedge \overline{x}<\underline{y} \wedge \overline{y}=\underline{y}]$

$\psi(\underline{s},\overline{s}) = [\exists i,j\in C, \underline{x},\underline{y},\underline{q},\overline{x},\overline{y},\overline{q}\in \mathbb{Z} | \underline{s}=<i,(\underline{x},\underline{y},\underline{q})> \wedge \overline{s}=<j,(\overline{x},\overline{y},\overline{q})> \wedge \overline{\psi}(\underline{x},\underline{y},\underline{q},\overline{x},\overline{y},\overline{q})]$

then when we said that the program was partially correct we meant :

$\forall \underline{s},\overline{s}\in S, [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$

4. INVARIANCE PROPERTIES OF PROGRAMS

Partial correctness is a special case of invariance property. More generally, if $\varepsilon\in(S\rightarrow\{tt,ff\})$ characterizes initial states, $\sigma\in(S\rightarrow\{tt,ff\})$ characterizes final states, $t\in((SxS)\rightarrow\{tt,ff\})$ is a transition relation then $\psi\in((SxS)\rightarrow\{tt,ff\})$ is said to be invariant for t with respect to ε and σ iff by definition :

$[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$

4.1 PARTIAL CORRECTNESS

Partial correctness is an invariance property where ε characterizes entry states, σ characterizes exit states and ψ is the relation between entry and exit states. Notice that the fact that an exit state $\overline{s}$ can be reached when execution is started with an entry state $\underline{s}$ is an hypothesis which must be true for $\psi(\underline{s},\overline{s})$ to hold, so that termination is not implied. This definition of partial correctness also covers the cases when the program is not partially correct for all initial values of the variables. For example we might have chosen :

$\overline{\psi}(\underline{x},\underline{y},\underline{q},\overline{x},\overline{y},\overline{q}) = [(\underline{x}\geq 0 \wedge \underline{y}>0)\Rightarrow(\underline{x}=\overline{q}*\underline{y}+\overline{x} \wedge 0\leq\overline{x}<\underline{y} \wedge \overline{y}=\underline{y})]$

4.2 NON-TERMINATION

Non-termination is also an invariance property where ε characterizes entry states, σ is identically true, $\psi(\underline{s},\overline{s})$ is true iff $\overline{s}$ is not an exit state.

For example, we can choose for our program

$$\varepsilon(\underline{s}) = [\exists\, \underline{x},\underline{y},\underline{q} \in \mathbb{Z} \mid \underline{s}=\langle 1,(\underline{x},\underline{y},\underline{q})\rangle \wedge (\underline{x}\geq\underline{y}) \wedge (\underline{y}=0)]$$
$$\sigma(\overline{s}) = \text{true}$$
$$\psi(\underline{s},\overline{s}) = \overline{[\exists i\in C,\ m\in M \mid \overline{s}=\langle i,m\rangle \wedge i\neq 6]}$$

When we claim that the program does not terminate when Y is initially zero and X positive, we mean :

$$[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$$

that is no exit state can be reached during execution.

4.3 CLEAN BEHAVIOR

If our program is to be implemented on a machine then integer variables can only have values between two bounds say "min" and "max". The transition relation for this program would then be :

$$\langle 1,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 2,(x,y,0)\rangle \quad \text{iff } \min\leq 0\leq\max$$
$$\langle 2,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 3,(x,y,q)\rangle \quad \text{iff } x\geq y$$
$$\langle 2,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 6,(x,y,q)\rangle \quad \text{iff } x<y$$
$$\langle 3,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 4,(x,y,q+1)\rangle \quad \text{iff } \min\leq q+1\leq\max$$
$$\langle 4,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 5,(x-y,y,q)\rangle \quad \text{iff } \min\leq x-y\leq\max$$
$$\langle 5,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 3,(x,y,q)\rangle \quad \text{iff } x\geq y$$
$$\langle 5,(x,y,q)\rangle \overset{t}{\rightarrow} \langle 6,(x,y,q)\rangle \quad \text{iff } x<y$$

where x,y,q are universally quantified over integers included between min and max.

When we claim that execution of the program does not lead to a run-time error when the initial value of X is positive and that of Y is strictly positive we mean :

$$[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$$

where

$$\varepsilon(\underline{s}) = [\exists \underline{x},\underline{y},\underline{q}\in \mathbb{Z} \mid \underline{s}=\langle 1,(\underline{x},\underline{y},\underline{q})\rangle \wedge 0\leq\underline{x}\leq\max \wedge 0<\underline{y}\leq\max]$$
$$\sigma(\overline{s}) = [\exists i\in C,\ m\in M \mid \overline{s}=\langle i,m\rangle \wedge i\neq 6]$$
$$\psi(\underline{s},\overline{s}) = [\exists s'\in S \mid t(\overline{s},s')]$$

that is whenever a state is reached during execution, which is not an exit state, then a well-defined computation step is possible.

4.4 GLOBAL INVARIANCE

When we claim that execution of the program leaves $(y=\underline{y})$ invariant, we mean that no operation of the program can modify the truth of this predicate, that is

$$[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$$

where

$$\varepsilon(\underline{s}) = [\exists m \in \mathbb{N} | \underline{s}=<1,m>]$$
$$\sigma(\overline{s}) = \underline{\text{true}}$$
$$\psi(\underline{s},\overline{s}) = [\exists j \in C,\ \underline{x},\underline{y},\underline{q},\overline{x},\overline{y},\overline{q} \in \mathbb{Z} | \underline{s}=<1,(\underline{x},\underline{y},\underline{q})> \wedge \overline{s}=<j,(\overline{x},\overline{y},\overline{q})> \wedge \overline{y}=\underline{y}]$$

5. DISCOVERING THE INDUCTION PRINCIPLE UNDERLYING AN INVARIANCE PROOF METHOD

We have given an abstract definition of invariance properties of programs. We now want to single out a few properties of invariance proof methods in order to omit from consideration a lot of details which can only obscure our understanding of these proof methods.

We start from the partial correctness proof of the program considered at paragraph 2.3 using the relational forward deductive positive method (*I*).

Our first abstraction consists in noticing that the intermediate assertions P_i associated with program point i, i=1,...,6, can be understood as a relation I on states. We had :

$$P_1(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y}]$$
$$P_2(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y} \wedge q=0]$$
$$P_3(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=q*\underline{y}+x]$$
$$P_4(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=(q-1)*\underline{y}+x]$$
$$P_5(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=q*\underline{y}+x]$$
$$P_6(\underline{x},\underline{y},x,y,q) = [\underline{x}=q*\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}]$$

so that $I \in ((S \times S) \to \{tt,ff\})$ is

$$I(\underline{s},s) = [\exists j \in C,\ \underline{x},\underline{y},\underline{q},x,y,q \in \mathbb{Z} | \underline{s}=<1,(\underline{x},\underline{y},\underline{q})> \wedge s=<j,(x,y,q)> \wedge P_j(\underline{x},\underline{y},x,y,q)]$$

Our second abstraction consists in understanding the verification conditions on the P_i, i=1,...,6 in term of equivalent verification conditions involving I and our abstract definitions of the program and its partial correctness (that is in term of $S,t,\varepsilon,\sigma,\psi$ as defined at paragraph 3).

- The first verification condition was :

$$[x=\underline{x} \wedge y=\underline{y}] \Rightarrow P_1(\underline{x},\underline{y},x,y,q)$$

that is $P_1(\underline{x},\underline{y},\underline{x},\underline{y},\underline{q})$ which is equivalent to $\forall \underline{s},\ \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s})$ since $[\exists(\underline{x},\underline{y},\underline{q}) \in M \mid \underline{s} = <1,(\underline{x},\underline{y},\underline{q})>] \Rightarrow I(\underline{s},\underline{s})$ is equivalent to $I(<1,(\underline{x},\underline{y},\underline{q})>,<1,(\underline{x},\underline{y},\underline{q})>)$ that is :

$$[\exists j \in C \mid j=1 \wedge P_j(\underline{x},\underline{y},\underline{x},\underline{y},\underline{q})] = P_1(\underline{x},\underline{y},\underline{x},\underline{y},\underline{q})$$

- The verification conditions 2 to 6 :

$$[\exists q' \mid P_1(\underline{x},\underline{y},x,y,q') \wedge q=0] \Rightarrow P_2(\underline{x},\underline{y},x,y,q)$$
$$[(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge (x \geq y)] \Rightarrow P_3(\underline{x},\underline{y},x,y,q)$$
$$[\exists q' \mid P_3(\underline{x},\underline{y},x,y,q') \wedge q=q'+1] \Rightarrow P_4(\underline{x},\underline{y},x,y,q)$$
$$[\exists x' \mid P_4(\underline{x},\underline{y},x',y,q) \wedge x=x'-y] \Rightarrow P_5(\underline{x},\underline{y},x,y,q)$$
$$[(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge (x<y)] \Rightarrow P_6(\underline{x},\underline{y},x,y,q)$$

are equivalent to :

$$[\exists x',y',q' \mid P_1(\underline{x},\underline{y},x',y',q') \wedge x=x' \wedge y=y' \wedge q=0] \Rightarrow P_2(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_2(\underline{x},\underline{y},x',y',q') \wedge (x \geq y') \wedge x=x' \wedge y=y' \wedge q=q'] \Rightarrow P_3(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_5(\underline{x},\underline{y},x',y',q') \wedge (x \geq y') \wedge x=x' \wedge y=y' \wedge q=q'] \Rightarrow P_3(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_3(\underline{x},\underline{y},x',y',q') \wedge x=x' \wedge y=y' \wedge q=q'+1] \Rightarrow P_4(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_4(\underline{x},\underline{y},x',y',q') \wedge x=x'-y' \wedge y=y' \wedge q=q'] \Rightarrow P_5(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_2(\underline{x},\underline{y},x',y',q') \wedge (x<y') \wedge x=x' \wedge y=y' \wedge q=q'] \Rightarrow P_6(\underline{x},\underline{y},x,y,q)$$
$$[\exists x',y',q' \mid P_5(\underline{x},\underline{y},x',y',q') \wedge (x<y') \wedge x=x' \wedge y=y' \wedge q=q'] \Rightarrow P_6(\underline{x},\underline{y},x,y,q)$$

Using I, t, $\underline{m}=(\underline{x},\underline{y},\underline{q})$, $\underline{m'}=(\underline{x'},\underline{y'},\underline{q'})$, and $m=(x,y,q)$ this can be written as :

$$[I(<1,\underline{m}>,<1,\underline{m'}>) \wedge t(<1,\underline{m'}>,<2,m>)] \Rightarrow I(<1,\underline{m}>,<2,m>)$$
$$[I(<1,\underline{m}>,<2,\underline{m'}>) \wedge t(<2,\underline{m'}>,<3,m>)] \Rightarrow I(<1,\underline{m}>,<3,m>)$$
$$[I(<1,\underline{m}>,<5,\underline{m'}>) \wedge t(<5,\underline{m'}>,<3,m>)] \Rightarrow I(<1,\underline{m}>,<3,m>)$$
$$[I(<1,\underline{m}>,<3,\underline{m'}>) \wedge t(<3,\underline{m'}>,<4,m>)] \Rightarrow I(<1,\underline{m}>,<4,m>)$$
$$[I(<1,\underline{m}>,<4,\underline{m'}>) \wedge t(<4,\underline{m'}>,<5,m>)] \Rightarrow I(<1,\underline{m}>,<5,m>)$$
$$[I(<1,\underline{m}>,<2,\underline{m'}>) \wedge t(<2,\underline{m'}>,<6,m>)] \Rightarrow I(<1,\underline{m}>,<2,m>)$$
$$[I(<1,\underline{m}>,<5,\underline{m'}>) \wedge t(<5,\underline{m'}>,<6,m>)] \Rightarrow I(<1,\underline{m}>,<6,m>)$$

that is

$$\forall c',c \in C,\ [I(<1,\underline{m}>,<c',\underline{m'}>) \wedge t(<c',\underline{m'}>,<c,m>)] \Rightarrow I(<1,\underline{m}>,<c,m>)$$

which is equivalent to :

$$\forall \underline{s},s',s,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},s') \wedge t(s',s)] \Rightarrow I(\underline{s},s)$$

- The last verification condition was :

$$P_6(\underline{x},\underline{y},\overline{x},\overline{y},\overline{q}) \Rightarrow [\underline{x}=\overline{q}*\underline{y}+\overline{x} \wedge \overline{x}<\underline{y} \wedge \overline{y}=\underline{y}]$$

that is :

$$I(<1,\underline{m}>,<6,\overline{m}>) \Rightarrow \psi(<1,\underline{m}>,<6,\overline{m}>)$$

or

$$\forall \underline{s},\overline{s},\ [\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$$

Therefore, we have shown that the proof method *I* essentially consist in discovering an invariant I and proving that :

$$[(\forall \underline{s}\in S,\ \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s}))$$
$$\wedge(\forall \underline{s},\underline{s}',\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},\underline{s}') \wedge t(\underline{s}',\overline{s})] \Rightarrow I(\underline{s},\overline{s}))$$
$$\wedge(\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s}))]$$

from which we conclude :

$$\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$$

This abstraction from an example will now allow us to explain in the same way the sixteen proof methods which have been considered for our introductory example and to generalize these proof methods to invariance properties other than partial correctness.

6. INDUCTION PRINCIPLES FOR INVARIANCE PROOFS AND THEIR RELATIONSHIPS

6.1 RELATIONAL FORWARD DEDUCTIVE POSITIVE METHOD (*I*)

The induction principle underlying proof method *I* has just been shown to be :

$$\begin{array}{l} [\exists I\in(S^2 \to \{tt,ff\}) \mid \forall \underline{s},\underline{s},\overline{s}\in S, \\ (I.\varepsilon) \quad \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s}) \\ (I.i) \quad \wedge \ [\exists \underline{s}'\in S \mid \varepsilon(\underline{s}) \wedge I(\underline{s},\underline{s}') \wedge t(\underline{s}',\underline{s})] \Rightarrow I(\underline{s},\underline{s}) \\ (I.\sigma) \quad \wedge \ [\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})] \\ \quad \Leftrightarrow \\ \quad [\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})] \end{array} \qquad (I)$$

The soundness proof ($\Rightarrow$) which was given at paragraph 2.3 for a particular example can easily be generalized. By recurrence on n, we first show that $[\forall \underline{s},s\in S,\forall n\geq 0,[\varepsilon(\underline{s}) \wedge t^n(\underline{s},s)] \Rightarrow I(\underline{s},s)]$. We use $(I.\varepsilon)$ for the basis n=0 and $(I.i)$ for the induction step n>0. From this lemma we conclude that $[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow [\varepsilon(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})]$ which according to $(I.\sigma)$ implies $\psi(\underline{s},\overline{s})$.

The completeness proof ($\Leftarrow$) is also very simple since we can choose $I(\underline{s},s) = t^*(\underline{s},s)$ so that $(I.\varepsilon)$ and $(I.i)$ follow from immediate properties of the reflexive transitive closure whereas $(I.\sigma)$ follows from the hypothesis $[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})$.

All other induction principles will be deduced from this one, using simple transformations.

6.2 RELATIONAL FORWARD PREDICTIVE POSITIVE METHOD ($\tilde{I}$)

The verification condition ($\tilde{I}$.i) :

$$[\forall \underline{s},s\in S,\ [\exists s'\in S \mid \varepsilon(\underline{s}) \wedge I(\underline{s},s') \wedge t(s',s)] \Rightarrow I(\underline{s},s)]$$

is equivalent to :

$$\begin{aligned}
&[\forall \underline{s},s',s\in S,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},s')] \Rightarrow [t(s',s) \Rightarrow I(\underline{s},s)]]\\
\Leftrightarrow\ &[\forall \underline{s},s'\in S,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},s')] \Rightarrow [\forall s\in S,\ t(s',s) \Rightarrow I(\underline{s},s)]]\\
\Leftrightarrow\ &[\forall \underline{s},s\in S,\ [\varepsilon(\underline{s}) \wedge I(\underline{s},s)] \Rightarrow \neg[\exists s'\in S \mid t(s,s') \wedge \neg I(\underline{s},s')]
\end{aligned}$$

Therefore from 6.1, we derive the equivalent induction principle $\tilde{I}$:

$$\begin{array}{ll}
 & [\exists I\in(S^2\rightarrow\{tt,ff\}) \mid \forall \underline{s},s,\overline{s}\in S,\\
(\tilde{I}.\varepsilon) & \quad \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s})\\
(\tilde{I}.i) & \wedge\ [\varepsilon(\overline{s}) \wedge I(\underline{s},s)] \Rightarrow \neg[\exists s'\in S \mid t(s,s') \wedge \neg I(\underline{s},s')]\\
(\tilde{I}.\sigma) & \wedge\ [\varepsilon(\overline{s}) \wedge I(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]\\
 & \Leftrightarrow\\
 & [\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]
\end{array} \qquad (\tilde{I})$$

6.3 RELATIONAL BACKWARD DEDUCTIVE POSITIVE METHOD ($\tilde{I}^{-1}$)

If we want to prove that :

$$[\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]$$

we can reason on inverse relations $\psi^{-1}(\overline{s},\underline{s})=\psi(\underline{s},\overline{s})$ and $(t^*(\overline{s},\underline{s}))^{-1}=t^*(\underline{s},\overline{s})$ and prove :

$$[\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge (t^*(\overline{s},\underline{s}))^{-1} \wedge \sigma(\overline{s})] \Rightarrow \psi^{-1}(\overline{s},\underline{s})]$$

Renaiming the dummy variables $\underline{s}$ and $\overline{s}$ respectively as $\overline{s}$ and $\underline{s}$ and using the commutativity of the conjunction $\wedge$ we get :

$$[\forall \overline{s},\underline{s}\in S,\ [\sigma(\underline{s}) \wedge (t^*(\underline{s},\overline{s}))^{-1} \wedge \varepsilon(\overline{s})] \Rightarrow \psi^{-1}(\underline{s},\overline{s})$$

Since $(t^*(\underline{s},\overline{s}))^{-1}=t^{-1*}(\underline{s},\overline{s})$ we can rewrite this as :

$$[\forall \underline{s},\overline{s}\in S,\ [\sigma(\underline{s}) \wedge t^{-1*}(\underline{s},\overline{s}) \wedge \varepsilon(\overline{s})] \Rightarrow \psi^{-1}(\underline{s},\overline{s})$$

This can be proved using induction principle I where $\varepsilon,t,\sigma,\psi$ are respectively chosen as $\sigma,t^{-1},\varepsilon,\psi^{-1}$ so that we get the verification conditions :

$$\begin{array}{l}
[\exists I\in(S^2\rightarrow\{tt,ff\}) \mid \forall \underline{s},s,\overline{s}\in S,\\
\quad \sigma(\underline{s}) \Rightarrow I(\underline{s},\underline{s})\\
\wedge\ [\exists s'\in S \mid \sigma(\overline{s}) \wedge I(\underline{s},s') \wedge t^{-1}(s',s)] \Rightarrow I(\underline{s},s)\\
\wedge\ [\sigma(\underline{s}) \wedge I(\underline{s},\overline{s}) \wedge \varepsilon(\overline{s})] \Rightarrow \psi^{-1}(\underline{s},\overline{s})]
\end{array}$$

Let J be the inverse I^{-1} of I. These verification conditions are equivalent to :

$$\begin{array}{l}
[\exists J\in(S^2\rightarrow\{tt,ff\}) \mid \forall \underline{s},s,\overline{s}\in S,\\
\quad \sigma(\underline{s}) \Rightarrow J(\underline{s},\underline{s})\\
\wedge\ [\exists \overline{s}'\in S \mid t^{-1}(s',s) \wedge J(s',\underline{s}) \wedge \sigma(\underline{s})] \Rightarrow J(s,\underline{s})\\
\wedge\ [\varepsilon(\overline{s}) \wedge J(\overline{s},\underline{s}) \wedge \sigma(\underline{s})] \Rightarrow \psi^{-1}(\underline{s},\overline{s})]
\end{array}$$

Renaming the dummy variables $\underline{s},\overline{s}$ respectively as $\overline{s},\underline{s}$, we get equivalently :

$$\begin{array}{l}[\exists J\in(S^2\to\{tt,ff\})\mid\forall\overline{s},\underline{s},\underline{s}\in S,\\ \quad \sigma(\overline{s})\Rightarrow J(\overline{s},\overline{s})\\ \quad\wedge\ [\exists\underline{s}'\in S\mid t^{-1}(\underline{s}',\underline{s})\wedge J(\underline{s}',\overline{s})\wedge\sigma(\overline{s})]\Rightarrow J(\underline{s},\overline{s})\\ \quad\wedge\ [\varepsilon(\underline{s})\wedge J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi^{-1}(\overline{s},\underline{s})]\end{array}$$

Using the definition of inverse relations, we have just proved that the proof method I^{-1} is sound and complete :

$$\begin{array}{ll}&[\exists J\in(S^2\to\{tt,ff\})\mid\forall\underline{s},\underline{s},\overline{s}\in S,\\ (I^{-1}.\sigma)&\quad\sigma(\overline{s})\Rightarrow J(\overline{s},\overline{s})\\ (I^{-1}.i)&\wedge\ [\exists\underline{s}'\in S\mid t(\underline{s},\underline{s}')\wedge J(\underline{s}',\overline{s})\wedge\sigma(\overline{s})]\Rightarrow J(\underline{s},\overline{s})\\ (I^{-1}.\varepsilon)&\wedge\ [\varepsilon(\underline{s})\wedge J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi(\underline{s},\overline{s})]\\ &\Leftrightarrow\\ &[\forall\underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s})\wedge t^{*}(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi(\underline{s},\overline{s})]\end{array}\qquad(I^{-1})$$

6.4 RELATIONAL BACKWARD PREDICTIVE POSITIVE METHOD ($\tilde{I^{-1}}$)

This proof method is equivalent to I^{-1} since

$$\begin{array}{ll}&[\forall\underline{s},\overline{s}\in S,\ [\exists\underline{s}'\in S\mid t(\underline{s},\underline{s}')\wedge J(\underline{s}',\overline{s})\wedge\sigma(\overline{s})]\Rightarrow J(\underline{s},\overline{s})]\\ \Leftrightarrow&[\forall\underline{s},\overline{s}\in S,\ [J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\neg[\exists\underline{s}'\in S\mid\neg J(\underline{s}',\overline{s})\wedge t(\underline{s}',\underline{s})]\end{array}$$

$$\begin{array}{ll}&[\exists J\in(S^2\to\{tt,ff\})\mid\forall\underline{s},\underline{s},\overline{s}\in S,\\ (\tilde{I^{-1}}.\sigma)&\quad\sigma(\overline{s})\Rightarrow J(\overline{s},\overline{s})\\ (\tilde{I^{-1}}.i)&\wedge\ [J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\neg[\exists\underline{s}'\in S\mid\neg J(\underline{s}',\overline{s})\wedge t(\underline{s}',\underline{s})]\\ (\tilde{I^{-1}}.\varepsilon)&\wedge\ [\varepsilon(\underline{s})\wedge J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi(\underline{s},\overline{s})]\\ &\Leftrightarrow\\ &[\forall\underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s})\wedge t^{*}(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi(\underline{s},\overline{s})]\end{array}\qquad(\tilde{I^{-1}})$$

6.5 RELATIONAL FORWARD DEDUCTIVE CONTRAPOSITIVE METHOD ($\overline{I^{-1}}$)

Using the property that $\neg\neg J=J$, we can rewrite the verification conditions I^{-1} as :

$$\begin{array}{l}[\exists J\in(S^2\to\{tt,ff\})\mid\forall\underline{s},\underline{s},\overline{s}\in S,\\ \quad[\varepsilon(\underline{s})\wedge\neg\neg J(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\psi(\underline{s},\overline{s})\\ \quad\wedge\ [\exists\underline{s}'\in S\mid t(\underline{s},\underline{s}')\wedge\neg\neg J(\underline{s}',\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\neg\neg J(\underline{s},\overline{s})\\ \quad\wedge\ \sigma(\overline{s})\Rightarrow\neg\neg J(\overline{s},\overline{s})]\end{array}$$

If we let $\overline{J}$ be $\neg J$ in the above condition and use the fact that $(P\Rightarrow Q)$ iff $(\neg Q\Rightarrow\neg P)$ we notice that it is equivalent to :

$$\begin{array}{l}[\exists\overline{J}\in(S^2\to\{tt,ff\})\mid\forall\underline{s},\underline{s}',\overline{s}\in S,\\ \quad[\varepsilon(\underline{s})\wedge\neg\psi(\underline{s},\overline{s})\wedge\sigma(\overline{s})]\Rightarrow\overline{J}(\underline{s},\overline{s})\\ \quad\wedge\ [\exists\underline{s}\in S\mid\overline{J}(\underline{s},\overline{s})\wedge t(\underline{s},\underline{s}')\wedge\sigma(\overline{s})]\Rightarrow\overline{J}(\underline{s}',\overline{s})\\ \quad\wedge\ \sigma(\overline{s})\Rightarrow\neg\overline{J}(\overline{s},\overline{s})]\end{array}$$

from which we conclude that the induction principle $\overline{I^{-1}}$ is equivalent to I^{-1}, hence sound and complete :

$$
\begin{array}{ll}
 & [\exists \overline{J} \in (S^2 \to \{tt,ff\}) \mid \forall \underline{s},s,\overline{s} \in S, \\
(\overline{I^{-1}}.\varepsilon) & \quad [\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \overline{J}(\underline{s},\overline{s}) \\
(\overline{I^{-1}}.i) & \wedge \quad [\exists \overline{s'} \in S \mid \overline{J}(\overline{s'},\overline{s}) \wedge t(s',s) \wedge \sigma(\overline{s})] \Rightarrow \overline{J}(s,\overline{s}) \\
(I^{-1}.\sigma) & \quad \sigma(\overline{s}) \Rightarrow \neg\overline{J}(\overline{s},\overline{s})] \\
 & \Leftrightarrow \\
 & [\forall \underline{s},\overline{s} \in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]
\end{array}
\qquad (\overline{I^{-1}})
$$

6.6 RELATIONAL FORWARD PREDICTIVE CONTRAPOSITIVE METHOD ($\widetilde{\overline{I^{-1}}}$)

The deductive verification condition $(\overline{I^{-1}}.i)$ can be given a predictive form using the transformation of $[[\exists x' \mid P(x') \wedge t(x',x)] \Rightarrow Q(x)]$ into the equivalent form $[P(x) \Rightarrow \neg[\exists x' \mid t(x,x') \wedge \neg Q(x')]]$:

$$
\begin{array}{ll}
 & [\exists \overline{J} \in (S^2 \to \{tt,ff\}) \mid \forall \underline{s},s,\overline{s} \in S, \\
(\widetilde{\overline{I^{-1}}}.\varepsilon) & \quad [\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \overline{J}(\underline{s},\overline{s}) \\
(\widetilde{\overline{I^{-1}}}.i) & \wedge \quad [\overline{J}(s,\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \neg[\exists s' \in S \mid t(s,s') \wedge \neg\overline{J}(s',\overline{s})] \\
(\widetilde{\overline{I^{-1}}}.\sigma) & \quad \sigma(\overline{s}) \Rightarrow \neg\overline{J}(\overline{s},\overline{s})] \\
 & \Leftrightarrow \\
 & [\forall \underline{s},\overline{s} \in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]
\end{array}
\qquad (\widetilde{\overline{I^{-1}}})
$$

6.7 RELATIONAL BACKWARD DEDUCTIVE CONTRAPOSITIVE METHOD ($\overline{I}$)

The same way that we obtained the contrapositive version $\overline{I^{-1}}$ of I^{-1}, we can obtain the contrapositive version $\overline{I}$ of I which leads to :

$$
\begin{array}{ll}
 & [\exists \overline{I} \in (S^2 \to \{tt,ff\}) \mid \forall \underline{s},s,\overline{s} \in S, \\
(\overline{I}.\sigma) & \quad [\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \overline{I}(\underline{s},\overline{s}) \\
(\overline{I}.i) & \wedge \quad [\exists \overline{s'} \in S \mid \varepsilon(\underline{s}) \wedge t(s,s') \wedge \overline{I}(\underline{s},\overline{s'})] \Rightarrow \overline{I}(\underline{s},s) \\
(\overline{I}.\varepsilon) & \quad \varepsilon(\underline{s}) \Rightarrow \neg\overline{I}(\underline{s},\underline{s})] \\
 & \Leftrightarrow \\
 & [\forall \underline{s},\overline{s} \in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]
\end{array}
\qquad (\overline{I})
$$

6.8 RELATIONAL BACKWARD PREDICTIVE CONTRAPOSITIVE METHOD ($\widetilde{\overline{I}}$)

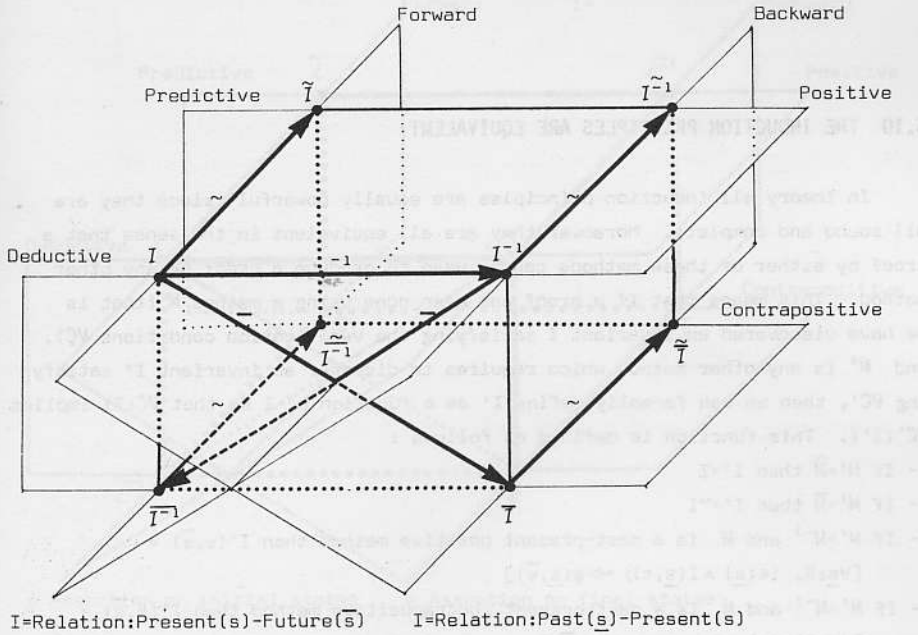
The predictive form of the above induction principle is :

$$
\begin{array}{ll}
 & [\exists \overline{I} \in (S^2 \to \{tt,ff\}) \mid \forall \underline{s},\underline{s},\overline{s} \in S, \\
(\tilde{\overline{I}}.\sigma) & \quad [\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \overline{I}(\underline{s},\overline{s}) \\
(\tilde{\overline{I}}.i) & \wedge \quad [\varepsilon(\underline{s}) \wedge \overline{I}(\underline{s},\underline{s})] \Rightarrow \neg[\exists \underline{s}' \in S \mid \neg\overline{I}(\underline{s},\underline{s}') \wedge t(\underline{s}',\underline{s})] \\
(\tilde{\overline{I}}.\varepsilon) & \wedge \quad \varepsilon(\underline{s}) \Rightarrow \neg\overline{I}(\underline{s},\underline{s})] \\
 & \Longleftrightarrow \\
 & [\forall \underline{s},\overline{s} \in S,\ [\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]
\end{array}
\qquad (\tilde{\overline{I}})
$$

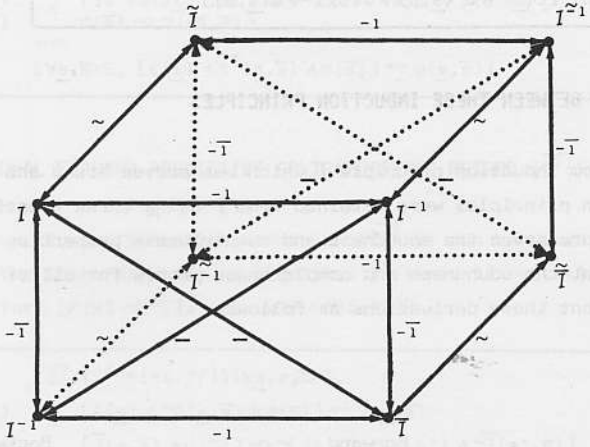
6.9 RELATIONSHIPS BETWEEN THESE INDUCTION PRINCIPLES

We started from induction principle I which was proved sound and complete. All other induction principles were obtained from I using three transformations $\sim$, $^{-1}$ and $-$ which preserved the soundness and completeness properties so that we had not to repeat the soundness and completeness proofs for all of them.

We can represent these derivations as follows :



The reader can check that further applications of the transformations $\sim$, -1 or $-$ to the above induction principles do not lead to a new induction principle. More generally, the following diagram commutes :



6.10 THE INDUCTION PRINCIPLES ARE EQUIVALENT

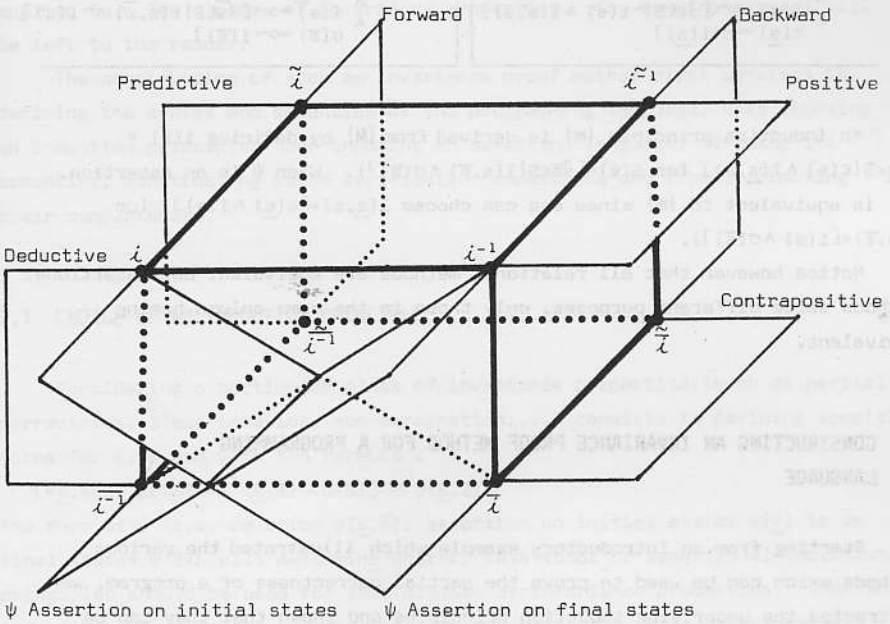
In theory all induction principles are equally powerful since they are all sound and complete. Moreover they are all equivalent in the sense that a proof by either of these methods can be used to produce a proof by any other method. This means that if a proof has been done using a method M (that is we have discovered an invariant I satisfying the verification conditions VC), and M' is any other method which requires to discover an invariant I' satisfying VC', then we can formally define I' as a function of I so that $VC(I)$ implies $VC'(I')$. This function is defined as follows :

- If $M'=\tilde{M}$ then $I'=I$
- If $M'=\overline{M}$ then $I'=\neg I$
- If $M'=M^{-1}$ and M is a past-present positive method then $I'(s,\overline{s}) = [\forall \underline{s} \in S,\ (\varepsilon(\underline{s}) \wedge I(\underline{s},s)) \Rightarrow \psi(\underline{s},\overline{s})]$
- If $M'=M^{-1}$ and M is a past-present contrapositive method then $I'(s,\overline{s}) = [\exists \underline{s} \in S \mid \varepsilon(\underline{s}) \wedge I(\underline{s},s) \wedge \neg\psi(\underline{s},\overline{s})]$

- If $M'=M^{-1}$ and M is a present-future positive method then $I'(\underline{s},s) = [\forall \overline{s} \in S,\ (I(s,\overline{s}) \wedge \sigma(\overline{s})) \Rightarrow \psi(\underline{s},\overline{s})]$
- If $M'=M^{-1}$ and M is a present-future contrapositive method then $I'(\underline{s},s) = [\exists \overline{s} \in S \mid I(s,\overline{s}) \wedge \sigma(\overline{s}) \wedge \neg\psi(\underline{s},\overline{s})]$

6.11 ASSERTIONAL INDUCTION PRINCIPLES

When the relation ψ which has to be proved invariant is an assertion on initial states or on final states, then the inductive invariant I which is used in the above induction principles does not need to be a relation and can be chosen as an assertion (see examples 2.1, 2.2, 2.7, 2.8). We can classify the corresponding induction principles as follows :



We have summarized these induction principles in the following table :

$\forall \underline{s},\overline{s}\in S,[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})$	$\forall \underline{s},\overline{s},[\varepsilon(\underline{s}) \wedge t^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s})$
$\Longleftrightarrow$	$\Longleftrightarrow$
$(\mathcal{i})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $\varepsilon(\underline{s}) \Rightarrow i(\underline{s})$ $\wedge$ $[\forall \overline{s}'\in S\mid i(\overline{s}') \wedge t(s',s)] \Rightarrow i(s)$ $\wedge$ $[i(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$	$(\mathcal{i}^{-1})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $\sigma(\overline{s}) \Rightarrow i(\overline{s})$ $\wedge$ $[\exists \underline{s}'\in S\mid t(s,s') \wedge i(s')] \Rightarrow i(s)$ $\wedge$ $[\varepsilon(\underline{s}) \wedge i(\underline{s})] \Rightarrow \psi(\underline{s})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$(\tilde{\mathcal{i}})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $\varepsilon(\underline{s}) \Rightarrow i(\underline{s})$ $\wedge$ $i(\overline{s}) \Rightarrow \neg[\exists \overline{s}'\in S\mid t(s,s') \wedge \neg i(s')]$ $\wedge$ $[i(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$	$(\tilde{\mathcal{i}}^{-1})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $\sigma(\overline{s}) \Rightarrow i(\overline{s})$ $\wedge$ $i(s) \Rightarrow \neg[\exists \underline{s}'\in S\mid \neg i(s') \wedge t(s',s)]$ $\wedge$ $[\varepsilon(\underline{s}) \wedge i(\underline{s})] \Rightarrow \psi(\underline{s})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$(\overline{\mathcal{i}})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $[\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow i(\overline{s})$ $\wedge$ $[\exists \overline{s}'\in S\mid t(s,s') \wedge i(s')] \Rightarrow i(s)$ $\wedge$ $\varepsilon(\underline{s}) \Rightarrow \neg i(\underline{s},\underline{s})]$	$(\overline{\mathcal{i}^{-1}})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $[\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s})] \Rightarrow i(\underline{s})$ $\wedge$ $[\exists \underline{s}'\in S\mid i(s') \wedge t(s',s)] \Rightarrow i(s)$ $\wedge$ $\sigma(\overline{s}) \Rightarrow \neg i(\overline{s})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$(\tilde{\overline{\mathcal{i}}})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $[\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow i(\overline{s})$ $\wedge$ $i(s) \Rightarrow \neg[\exists \underline{s}'\in S\mid \neg i(s') \wedge t(s',s)]$ $\wedge$ $\varepsilon(\underline{s}) \Rightarrow \neg i(\underline{s})]$	$(\tilde{\overline{\mathcal{i}^{-1}}})$ $[\exists i\in(S\rightarrow\{tt,ff\})\mid \forall \underline{s},s,\overline{s}\in S,$ $[\varepsilon(\underline{s}) \wedge \neg\psi(\underline{s})] \Rightarrow i(\underline{s})$ $\wedge$ $i(s) \Rightarrow \neg[\exists \overline{s}'\in S\mid t(s,s') \wedge \neg i(s')]$ $\wedge$ $\sigma(\overline{s}) \Rightarrow \neg i(\overline{s})]$

An induction principle (m) is derived from (M) by defining $i(s) = [\exists \underline{s}\in S\mid \varepsilon(\underline{s}) \wedge I(\underline{s},s)]$ (or $i(s)=[\exists \overline{s}\in S\mid I(s,\overline{s}) \wedge \sigma(\overline{s})]$). When ψ is an assertion, (M) is equivalent to (m) since one can choose $I(\underline{s},s)=[\varepsilon(\underline{s}) \wedge i(s)]$ (or $I(s,\overline{s})=[i(s) \wedge \sigma(\overline{s})]$).

Notice however that all relational methods are equivalent but assertional methods serve different purposes, only those in the same column beeing equivalent.

7. CONSTRUCTING AN INVARIANCE PROOF METHOD FOR A PROGRAMMING LANGUAGE

Starting from an introductory example which illustrated the various methods which can be used to prove the partial correctness of a program, we abstracted the underlying induction principles and shown that they can be used for proving any invariance property of programs.

We now study the symmetrical problem that is starting from an abstract induction principle, how can we derive the corresponding method for proving invariance properties of programs written in a particular programming language?

Since we want to design invariance proof methods for a programming language but not only for a particular program, and for each program P the proof method will clearly depend upon the corresponding set of states $S[\![P]\!]$ and transition relation $t[\![P]\!]$, we will have to proceed by induction on the syntax of programs. Except for this generalization, the process will exactly be the reverse of the one illustrated at paragraph 5. Since it is not possible to consider all programming language features in such a short paper, we will only pay attention to while-programs with simple variables. It is out of the question to examine in detail all possible invariance properties and all sixteen induction principles for that language since their combinations lead to too many different proof methods. Instead, we will give a mathematically constructive method for designing all these invariance proof methods. Our approach will be illustrated for a few induction principles applied to a few classes of invariance properties. The remaining cases will be left to the reader.

The construction of such an invariance proof method first consists in defining the syntax and semantics of the programming language, next choosing an induction principle, then choosing an assertion language, defining its semantics, constructing sound verification conditions and finally checking their completeness.

7.1 CHOICE OF AN INDUCTION PRINCIPLE

Considering a particular class of invariance properties (such as partial correctness, clean-behavior, non-termination,...) consists in defining special forms for ε, σ and ψ in the formula :

$$[\forall \underline{s},\overline{s}\in S,\ [\varepsilon(\underline{s}) \wedge t^{*}(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]$$

The form of ψ (i.e. relation $\psi(\underline{s},\overline{s})$, assertion on initial states $\psi(\underline{s})$ or on final states $\psi(\overline{s})$) will determine whether relational or assertional induction principles should be used for that classes of invariance properties. Then an induction principle can be selected (see 6.9 and 6.11), the choice beeing between forward and backward inductions, positive or contrapositive inductive invariants and deductive or predictive verification conditions.

7.2 CHOICE OF AN ASSERTION LANGUAGE

Once an induction principle has been chosen, the verification condition for a program P will be of the form :

$[\exists I \in A[\![P]\!] \mid VC[\![P]\!](\varepsilon,\sigma)(\psi)(I)]$

where the assertion language $A[\![P]\!]$ is $(S[\![P]\!]^2 \to \{tt,ff\})$ for relational induction principles and $(S[\![P]\!] \to \{tt,ff\})$ for assertional ones and the verification condition $VC[\![P]\!]$ is a conjunction of three conditions depending upon $t[\![P]\!]$, ε, σ and ψ.

In practice, the inductive invariant I on the program states will not be expressed as an element of $A[\![P]\!]$. The proof method has to provide for a way of expressing I in an equivalent but more convenient form. Let $\tilde{A}[\![P]\!]$ be the set of such equivalent forms $\tilde{I}$ of I.

Example : Coming back at paragraph 5, we had for the program P :

```
1:
2: Q:=0;
3: while X≥Y do
4:     Q:=Q+1;
5:     X:=X-Y;
6: od;
```

C = {1,...,6} contrul states
M = {X,Y,Q} → ℤ memory states
S = C × M states

An invariant $I \in A[\![P]\!] = (S^2 \to \{tt,ff\})$ was expressed as a vector $\tilde{I}=(P_1,\ldots,P_6)$ where $P_i \in (\mathbb{Z}^5 \to \{tt,ff\})$, i=1,...,6. Therefore $\tilde{A}[\![P]\!] = \prod_{c \in C} (\mathbb{Z}^5 \to \{tt,ff\})$. *End of Example.*

7.3 DEFINITION OF THE MEANING OF THE ASSERTION LANGUAGE

The meaning of $\tilde{I} \in \tilde{A}[\![P]\!]$ can be defined by giving the equivalent $I \in A[\![P]\!]$, that is the equivalent assertion or relation on program states. For that purpose we introduce a pair of functions $\rho[\![P]\!] \in (\tilde{A}[\![P]\!] \to A[\![P]\!])$ and $\tilde{\rho}[\![P]\!] \in (A[\![P]\!] \to \tilde{A}[\![P]\!])$ such that $\rho[\![P]\!](\tilde{I})$ is the meaning of $\tilde{I} \in \tilde{A}[\![P]\!]$ and $\tilde{\rho}[\![P]\!](I)$ is the representation of $I \in A[\![P]\!]$.

Example (Cont'd) : The meaning of $\tilde{I}=(P_1,\ldots,P_6)$ is $I=\rho(\tilde{I})$ such that $I(\underline{s},s) = [\exists i \in C, \underline{x},\underline{y},\underline{q},x,y,q \in \mathbb{Z} \mid \underline{s}=<1,(\underline{x},\underline{y},\underline{q})> \wedge s=<i,(x,y,q)> \wedge P_i(\underline{x},\underline{y},x,y,q)]$.

This formalizes the fact that $P_i(\underline{x},\underline{y},x,y,q)$ is true between the initial values $\underline{x},\underline{y}$ and current values x,y,q of the program variables X,Y,Q when control reaches program point i.

Reciprocally, an invariant $I \in A[\![P]\!] = (S^2 \to \{tt,ff\})$ can be represented by $\tilde{\rho}(I) = (P_1,\ldots,P_6) \in \tilde{A}[\![P]\!]$ such that for $i=1,\ldots,6$, $P_i(\underline{x},\underline{y},x,y,q) = [\exists \underline{q} \in \mathbb{Z} \mid I(\langle 1,(\underline{x},\underline{y},\underline{q})\rangle,\langle i,(x,y,q)\rangle)]$. *End of Example.*

In Cousot & Cousot[80b], we study properties of the pair $(\tilde{\rho},\rho)$ which ensure that the definition of the semantics of $\tilde{A}$ with respect to A will be non-ambiguous, expressive, etc.

7.4 DESIGN OF A SOUND INVARIANCE PROOF METHOD

The induction principles studied at paragraph-6 are of the form :

$$[\exists I \in A[\![P]\!] \mid VC[\![P]\!](\varepsilon,\sigma)(\psi)(I)]$$
$$\Longleftrightarrow$$
$$[\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]$$

In practice, the inductive invariant $I \in A[\![P]\!]$ is expressed by a more convenient $\tilde{I} \in \tilde{A}[\![P]\!]$ so that the proof is of the form :

$$[\exists \tilde{I} \in \tilde{A}[\![P]\!] \mid \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I})]$$

where $\tilde{VC}$ is the verification condition corresponding to VC. This proof method is sound if and only if :

$$[\exists \tilde{I} \in \tilde{A}[\![P]\!] \mid \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I})] \Rightarrow [\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})]$$

or equivalently :

$$[\exists \tilde{I} \in \tilde{A}[\![P]\!] \mid \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I})] \Rightarrow [\exists I \in A[\![P]\!] \mid VC[\![P]\!](\varepsilon,\sigma)(\psi)(I)]$$

If moreover, we require I to be the meaning of $\tilde{I}$, then a sufficient soundness condition is :

$$\forall \tilde{I} \in \tilde{A}[\![P]\!], \quad \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I}) \Rightarrow VC[\![P]\!](\varepsilon,\sigma)(\psi)(\rho[\![P]\!](\tilde{I}))$$

Therefore $\tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)$ can be chosen as $VC[\![P]\!](\varepsilon,\sigma)(\psi) \circ \rho[\![P]\!]$. By algebraic manipulations this condition can be expressed as a conjunction of simpler conditions defined by induction on the syntax of program P. Moreover algebraic simplifications are allowed since implication and not equality is required.

7.5 VERIFICATION OF THE COMPLETENESS OF THE INVARIANCE PROOF METHOD

Once a proof method has been designed, i.e. $\tilde{A}[\![P]\!]$ and $\tilde{VC}[\![P]\!]$ have been defined for all programs P in the considered language, we can check that the method is complete, that is :

$$[\forall \underline{s},\overline{s}\in S[\![P]\!],[\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\underline{s},\overline{s})] \Rightarrow [\exists \tilde{I}\in \tilde{A}[\![P]\!] \mid \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I})]$$

or equivalently :

$$[\exists I\in A[\![P]\!] \mid VC[\![P]\!](\varepsilon,\sigma)(\psi)(I)] \Rightarrow [\exists \tilde{I}\in \tilde{A}[\![P]\!] \mid \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I})]$$

If moreover we require $\tilde{I}$ to be the representation of I, we obtain the following sufficient completeness condition :

$$\forall I\in A[\![P]\!],\ VC[\![P]\!](\varepsilon,\sigma)(\psi)(I) \Rightarrow \tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{\rho}[\![P]\!](I))$$

8. SYNTAX AND OPERATIONAL SEMANTICS OF THE EXAMPLE PROGRAMMING LANGUAGE

Before applying our opproach to a few examples, we have to define the programming language which will be used for these examples.

8.1 SYNTAX

A program consists of a list of sequentially executed commands. Commands can be null, assignment, conditional or iteration commands. Each command is preceded and followed by unique labels the only purpose of which is to designate program points.

Let L,V,E and B be respectively given sets of labels, simple variables, expressions and boolean expressions; $(L\cap V=\emptyset)$. The following context-free grammar then defines the set P of programs (command lists) and C of commands :

$P ::= L_1:C_1;\ldots;L_n:C_n;L_{n+1}:$ $\quad (n\geq 1)$

$C ::= \underline{\text{skip}} \mid V:=E \mid \underline{\text{if}}\ B\ \underline{\text{then}}\ P_1\ \underline{\text{else}}\ P_2\ \underline{\text{fi}} \mid \underline{\text{while}}\ B\ \underline{\text{do}}\ P\ \underline{\text{od}}$

Given a terminal string N deriving from the non-terminal N we write $N\equiv\alpha_1N_1\ldots\alpha_nN_n\alpha_{n+1}$ to state that there exist may be empty terminal strings α_i and non-empty terminal strings N_i such that N is of the form $\alpha_1N_1\ldots\alpha_nN_n\alpha_{n+1}$

and each string N_i derives from the non-terminal N_i (more formally we have $N \overset{*}{\rightarrow} \alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1} \overset{*}{\rightarrow} \alpha_1 N_1 \ldots \alpha_n N_n \alpha_{n+1} = N$).

For example, if labels are integers and P is the string 1:skip;2:skip;3: then $P \equiv \alpha 1{:}C;\beta$ is true since choosing α empty and β=2:skip;3: we have $P \overset{*}{\rightarrow}$ 1:C;2:skip;3: = $\alpha 1{:}C;\beta \overset{*}{\rightarrow} \alpha 1{:}$skip;$\beta$ = P.

This notation will be used to express context-sensitive properties of programs such as, for example, that labels can only appear once in a program :

$(\forall P \in \mathcal{P}, [P \equiv \alpha L_1 : \beta L_2 : \gamma] \Rightarrow [L_1 \neq L_2])$

8.2 OPERATIONAL SEMANTICS

8.2.1 States

The states <L,m> of a program $P \in \mathcal{P}$ consist of a control state L (i.e. a program point) and a memory state i.e. a partial function m which defines the value m(v) of the program variables v. This value, whenever it exists, belongs to some domain $\mathcal{D}$ which will be left unspecified. Formally, for a program $P \in \mathcal{P}$ we define :

$\mathbb{C}[\![P]\!] = \{L \in \mathcal{L} \mid P \equiv \alpha L{:}\beta\}$, the set of labels which appear in program P
$\mathbb{V}[\![P]\!] = \{v \in \mathcal{V} \mid P \equiv \alpha v \beta\}$, the set of variables which appear in program P
$\mathbb{M}[\![P]\!] = (\mathbb{V}[\![P]\!] \rightarrow \mathcal{D})$, the set of memory states
$\mathbb{S}[\![P]\!] = \mathbb{C}[\![P]\!] \times \mathbb{M}[\![P]\!]$, the set of states

8.2.2 Transition Relation

The transition relation $t[\![P]\!]$ between a state <L,m> of program P and its only (if any) possible successor <L',m'> is considered as a binary function on states with boolean result :

$t[\![P]\!] \in (\mathbb{S}[\![P]\!]^2 \rightarrow \{tt, ff\})$

<L',m'> is a successor of state <L,m> if and only if when executing the command designated by label L in memory state m, control goes to the program point designated by label L' (i.e. $tc[\![P]\!](L,m,L')$ holds) and memory state m is changed to $m' = tm[\![P]\!](L,m)$:

$t[\![P]\!](\langle L,m\rangle, \langle L',m'\rangle) = [\, tc[\![P]\!](L,m,L') \wedge (m' = tm[\![P]\!](L,m))\,]$

Memory states can only be changed by assignment commands v:=E. Let $\mathbb{E} \in (E \to (M[\![P]\!] \to \mathcal{D}))$ be a given semantic function, such that $\mathbb{E}[\![E]\!](m)$ is the value of expression E in memory state m. The effect of an assignment v:=E; is to evaluate expression E and assign its value to variable v. No other variable is modified since side-effects are disallowed :

$$tm[\![P]\!](L,m) = \underline{\text{if}}\ (P \Xi \alpha L : v := E ; \beta)\ \underline{\text{then}}\ m[v/\mathbb{E}[\![E]\!](m)]\ \underline{\text{else}}\ m$$

where $m[v/d](w)=m(w)$ when $w \in V[\![P]\!]-\{v\}$ and $m[v/d](v)=d$.

The transition relation $tc[\![P]\!](L,m,L')$ between a control state L and its successor L' in memory state m is defined by cases as follows :

$$tc[\![P]\!](L,m,L') =$$

$$\left[\begin{array}{l}
\left[\begin{array}{l}
\quad P \Xi \alpha L : \underline{\text{skip}} ; L' : \beta \\
\vee\ P \Xi \alpha L : \underline{\text{else}}\ P'\ \underline{\text{fi}} ; L' : \beta \\
\vee\ P \Xi \alpha L : \underline{\text{fi}} ; L' : \beta \\
\vee\ P \Xi \alpha L : v := E ; L' : \beta \wedge m \in dom(\mathbb{E}[\![E]\!])
\end{array}\right] \\
\vee \left[\begin{array}{l}
\left(\begin{array}{l}
\quad P \Xi \alpha L : \underline{\text{if}}\ B\ \underline{\text{then}}\ L' : \beta \\
\vee\ P \Xi \alpha L : \underline{\text{while}}\ B\ \underline{\text{do}}\ L' : \beta \\
\vee\ P \Xi \alpha \underline{\text{while}}\ B\ \underline{\text{do}}\ L' : C_1 ; \ldots ; L_n : C_n ; L : \underline{\text{od}} ; \beta
\end{array}\right) \\
\wedge\ (m \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](m) = tt)
\end{array}\right] \\
\vee \left[\begin{array}{l}
\left(\begin{array}{l}
\quad P \Xi \alpha L : \underline{\text{if}}\ B\ \underline{\text{then}}\ P'\ \underline{\text{else}}\ L' : \beta \\
\vee\ P \Xi \alpha L : \underline{\text{while}}\ B\ \underline{\text{do}}\ P'\ \underline{\text{od}} ; L' : \beta \\
\vee\ P \Xi \alpha \underline{\text{while}}\ B\ \underline{\text{do}}\ L_1 : C_1 ; \ldots ; L_n : C_n ; L : \underline{\text{od}} ; L' : \beta
\end{array}\right) \\
\wedge\ (m \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](m) = ff)
\end{array}\right]
\end{array}\right]$$

For example, if control is at point L before executing a while loop or after executing its body, then control goes to point L' designating the first command of its body if the test is well-defined and true, control exits the loop if the test is well-defined and false and program execution stops (at L which has no possible successor) if the test is ill-defined (that is its evaluation leads to a run-time error).

9. EXAMPLE : DESIGN OF A METHOD FOR PROVING NON-TERMINATION OF PROGRAMS

9.1 NON-TERMINATION AS AN INVARIANCE PROPERTY

Let $P \in \mathcal{P}$ be a program, and $\phi \in (M[\![P]\!] \to \{tt,ff\})$ be a characterization of the possible values of the program variables on entry. Program P does not properly terminate if and only if no reachable state during execution is an exit state, that is :

$$[\forall \underline{s},\overline{s} \in S[\![P]\!],\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$$

where

$$\varepsilon(\langle L,m\rangle) = [(P \equiv L:\alpha) \wedge \phi(m)]$$
$$\sigma(\langle L,m\rangle) = tt$$
$$\psi(\langle L,m\rangle) = [\neg(P \equiv \alpha L:)]$$

9.2 CHOICE OF AN INDUCTION PRINCIPLE

Since ψ is an assertion upon final states, paragraph 6.11 indicates that induction principles $(\mathcal{L})$, $(\widetilde{\mathcal{L}})$, $(\overline{\mathcal{L}})$ or $(\overline{\widetilde{\mathcal{L}}})$ can be used. We choose $(\overline{\widetilde{\mathcal{L}}})$ as example since this induction principle is not at all conventional. $(\overline{\widetilde{\mathcal{L}}})$ is of the form:

$$[\exists I \in A[\![P]\!] \mid VC[\![P]\!](\varepsilon,\sigma)(\psi)(I)]$$

where

$$A[\![P]\!] = (S[\![P]\!] \to \{tt,ff\})$$
$$VC[\![P]\!](\varepsilon,\sigma)(\psi)(I) = VC_\sigma[\![P]\!](\psi,\sigma)(I) \wedge VC_i[\![P]\!](I) \wedge VC_\varepsilon[\![P]\!](\varepsilon)(I)$$
$$VC_\sigma[\![P]\!](\psi,\sigma)(I) = [\forall \overline{s} \in S[\![P]\!], [\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow I(\overline{s})]$$
$$VC_i[\![P]\!](I) = [\forall s \in S[\![P]\!],\ I(s) \Rightarrow \neg[\exists s' \in S[\![P]\!] \mid \neg I(s') \wedge t[\![P]\!](s',s)]]$$
$$VC_\varepsilon[\![P]\!](\varepsilon)(I) = [\forall \underline{s} \in S[\![P]\!],\ \varepsilon(\underline{s}) \Rightarrow \neg I(\underline{s})]$$

9.3 CHOICE OF AN ASSERTION LANGUAGE

We decide to represent an assertion $I \in A[\![P]\!]$ by a vector $\widetilde{I}$ of assertions on memory states attached to each program point. Therefore

$$\widetilde{A}[\![P]\!] = \prod_{L \in C[\![P]\!]} (M[\![P]\!] \to \{tt,ff\})$$

The meaning of this vector of assertions is that the assertion $\widetilde{I}(L)(m)$ attached to program point L holds on the memory state m of all program states $\langle L,m\rangle$ having L as control state. More formally :

$$\rho[\![P]\!] \in \widetilde{A}[\![P]\!] \to A[\![P]\!]$$
$$\rho[\![P]\!](\widetilde{I})(\langle L,m\rangle) = \widetilde{I}(L)(m)$$

Reciprocally, an assertion $I \in A[\![P]\!]$ is represented by $\tilde{\rho}[\![P]\!](I) \in \tilde{A}[\![P]\!]$, that is a vector $\tilde{I}$ of assertions on memory states m such that :

$\forall L \in C[\![P]\!],\ \tilde{I}(L)(m) = I(\langle L,m\rangle)$

9.4 DESIGN OF SOUND VERIFICATION CONDITIONS

We have to define $\tilde{VC}[\![P]\!]$, such that :

$\tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I}) \Rightarrow VC[\![P]\!](\varepsilon,\sigma)(\psi)(\rho[\![P]\!](\tilde{I}))$

Since $VC[\![P]\!]$ is a conjunction of three verification conditions, we choose $\tilde{VC}[\![P]\!]$ to be of the same form, that is :

$\tilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\tilde{I}) = \tilde{VC}_\sigma[\![P]\!](\psi,\sigma)(\tilde{I}) \wedge \tilde{VC}_i[\![P]\!](\tilde{I}) \wedge \tilde{VC}_\varepsilon[\![P]\!](\varepsilon)(\tilde{I})$

In order to ensure the above soundness condition we will simply choose :

$\tilde{VC}_\sigma[\![P]\!](\psi,\sigma)(\tilde{I}) = VC_\sigma[\![P]\!](\psi,\sigma)(\rho[\![P]\!](\tilde{I}))$
$\tilde{VC}_i[\![P]\!](\tilde{I}) = VC_i[\![P]\!](\rho[\![P]\!](\tilde{I}))$
$\tilde{VC}_\varepsilon[\![P]\!](\varepsilon)(\tilde{I}) = VC_\varepsilon[\![P]\!](\varepsilon)(\rho[\![P]\!](\tilde{I}))$

9.4.1 Basis

$\tilde{VC}_\sigma[\![P]\!](\psi,\sigma)(\tilde{I})$

$= VC_\sigma[\![P]\!](\psi,\sigma)(\rho[\![P]\!](\tilde{I}))$
$= [\forall \overline{s} \in S[\![P]\!],\ [\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \rho[\![P]\!](\tilde{I})(\overline{s})]$

Using the fact that $S[\![P]\!] = C[\![P]\!] \times M[\![P]\!]$:

$= [\forall L \in C[\![P]\!],\ m \in M[\![P]\!],\ [\neg\psi(\langle L,m\rangle) \wedge \sigma(\langle L,m\rangle)] \Rightarrow \rho[\![P]\!](\tilde{I})(\langle L,m\rangle)]]$

Replacing ψ,σ and ρ by their definitions :

$= [\forall L \in C[\![P]\!],\ m \in M[\![P]\!],\ (P \Xi \alpha L:) \Rightarrow \tilde{I}(L)(m)]$

There is one and only one exit label for program P :

$= [(P \Xi \alpha L:) \wedge (\forall m \in M[\![P]\!],\ \tilde{I}(L)(m))]$

9.4.2 Induction Step

$\tilde{VC}_i[\![P]\!](\tilde{I})$

$= VC_i[\![P]\!](\rho[\![P]\!](\tilde{I}))$
$= [\forall s \in S[\![P]\!],\ \rho[\![P]\!](\tilde{I})(s) \Rightarrow \neg[\exists s' \in S[\![P]\!] \mid \neg\rho[\![P]\!](\tilde{I})(s') \wedge t[\![P]\!](s',s)]]$
$= [\forall s \in S[\![P]\!],\ \rho[\![P]\!](\tilde{I})(s) \Rightarrow [\forall s' \in S[\![P]\!],\ t[\![P]\!](s',s) \Rightarrow \rho[\![P]\!](\tilde{I})(s')]]$
$= [\forall L \in C[\![P]\!], m \in M[\![P]\!], \tilde{I}(L)(m) \Rightarrow [\forall L' \in C[\![P]\!], m' \in M[\![P]\!],$
$\qquad t[\![P]\!](\langle L',m'\rangle,\langle L,m\rangle) \Rightarrow \tilde{I}(L')(m')]]$
$= \bigwedge_{L \in C[\![P]\!]} [\forall m \in M[\![P]\!], \tilde{I}(L)(m) \Rightarrow [\forall L' \in C[\![P]\!], m' \in M[\![P]\!],$
$\qquad (tc[\![P]\!](L',m',L) \wedge m = tm[\![P]\!](L',m')) \Rightarrow \tilde{I}(L')(m')]]$

We have a conjunction of verification conditions, one for each program point $L \in C[\![P]\!]$, of the form :

$\tilde{I}(L)(m) \Rightarrow [\forall L' \in C[\![P]\!], m' \in M[\![P]\!], [\, tc[\![P]\!](L',m',L) \wedge m = tm[\![P]\!](L',m') \Rightarrow \tilde{I}(L')(m')\,]$

This condition can be further refined according to the various cases corresponding to the definition of $tc[\![P]\!]$:

(a) - If $P \equiv L:\alpha$ that is L designates the program entry point, then $\forall L' \in C[\![P]\!]$, $m' \in M[\![P]\!]$, $tc[\![P]\!](L',m',L)$ is false so that the verification condition is identically true.

(b) - If $P \equiv \alpha L'$: skip ; $L:\beta$ or $P \equiv \alpha L'$: else P' fi ; $L:\beta$ or $P \equiv \alpha L'$: fi ; $L:\beta$ then $tc[\![P]\!](L',m',L)$ is true and $tm[\![P]\!](L',m') = m'$ so that the verification condition can be simplified into :

$\tilde{I}(L)(m) \Rightarrow \tilde{I}(L')(m)$

(c) - If $P \equiv \alpha L'$: $v := E ; L:\beta$ then we must verify that :

$\tilde{I}(L)(m) \Rightarrow [\forall m' \in M[\![P]\!],\ m' \in dom(\mathbb{E}[\![E]\!]) \wedge m = m'[\, v/\mathbb{E}[\![E]\!](m')\,] \Rightarrow \tilde{I}(L')(m')\,]$

Notice that m' must be of the form $m[v/d]$ where $d \in \mathcal{D}$ is the value of v before the assignment. Therefore this condition can be simplified into :

$\tilde{I}(L)(m) \Rightarrow [\forall d \in \mathcal{D}, [m[v/d] \in dom(\mathbb{E}[\![E]\!]) \wedge m(v) = \mathbb{E}[\![E]\!](m[v/d])] \Rightarrow \tilde{I}(L')(m[v/d])]$

(d) - $P \equiv \alpha L'$: if B then $L:\beta$ or $P \equiv \alpha L'$: while B do $L:\beta$ or $P \equiv \alpha$ while B do $L:C_1;\ldots;L_n:C_n;L'$: od ; β

$\tilde{I}(L)(m) \Rightarrow [\,(m \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](m) = tt) \Rightarrow \tilde{I}(L')(m)\,]$

(e) - $P \equiv \alpha L'$: if B then P' else $L:\beta$ or $P \equiv \alpha L'$: while B do P' od ; $L:\beta$ or $P \equiv \alpha$ while B do $L_1:C_1;\ldots;L_n:C_n;L'$: od ; $L:\beta$

$\tilde{I}(L)(m) \Rightarrow [\,(m \in dom(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](m) = ff) \Rightarrow \tilde{I}(L')(m)\,]$

9.4.3 Contradiction

$$
\begin{aligned}
&\tilde{VC}_{\varepsilon}[\![P]\!](\varepsilon)(\tilde{I}) \\
&\quad = VC_{\varepsilon}[\![P]\!](\varepsilon)(\rho[\![P]\!](\tilde{I})) \\
&\quad = [\forall s \in S[\![P]\!], \varepsilon(s) \Rightarrow \neg\rho[\![P]\!](\tilde{I})(s)\,] \\
&\quad = [\forall L \in C[\![P]\!], m \in M[\![P]\!], [\,(P \equiv L:\alpha) \wedge \phi(m)\,] \Rightarrow \neg\tilde{I}(L)(m)\,] \\
&\quad = [\,(P \equiv L:\alpha) \wedge (\forall m \in M[\![P]\!], \phi(m) \Rightarrow \neg\tilde{I}(L)(m))\,]
\end{aligned}
$$

9.5 INFORMAL SUMMARY OF THE VERIFICATION CONDITIONS

Using informal mnemonic notations, we can recapitulate the above verification conditions as follows (P_i is the assertion on the program variables

attached to program point L_i) :

- Basis :

$\ldots:L_1$ $\qquad P_1$

- Induction step :

. Null command :

$\ldots L_1:\underline{skip};L_2:\ldots$ $\qquad P_2 \Rightarrow P_1$

. Assignment command :

$\ldots L_1:v:=E;L_2:\ldots$ $\qquad P_2 \Rightarrow [\forall v', (v=E|_v^{v'}) \Rightarrow P_1|_v^{v'}]$

. Conditional command :

$\ldots L_1:\underline{if}\ B\ \underline{then}$
$\quad L_2:\ldots$ $\qquad P_2 \Rightarrow [B \Rightarrow P_1]$
$\quad L_3:\ldots$
$\underline{else}$
$\quad L_4:\ldots$ $\qquad P_4 \Rightarrow [\neg B \Rightarrow P_1]$
$\quad L_5:\ldots$
$\underline{fi};$
$L_6:\ldots$ $\qquad P_6 \Rightarrow [P_3 \wedge P_5]$

. Iteration command :

$\ldots L_1:\underline{while}\ B\ \underline{do}$
$\quad L_2:\ldots$ $\qquad P_2 \Rightarrow [B \Rightarrow (P_1 \wedge P_3)]$
$\quad L_3:\ldots$
$\underline{od};$
$L_4:\ldots$ $\qquad P_4 \Rightarrow [\neg B \Rightarrow (P_1 \wedge P_3)]$

- Contradiction :

$L_1:\ldots$ $\qquad \phi \Rightarrow \neg P_1$

9.6 EXAMPLE OF PROOF USING THE DESIGNED METHOD

Let us prove that the following program :

```
1: Q:=0;
2: while X≥Y do
3:     Q:=Q+1;
4:     X:=X-Y;
5: od;
6:
```

does not terminate cleanly when the initial values $\underline{x},\underline{y}$ of variables X,Y are such that $\underline{x}\geq 0 \wedge \underline{y}=0$. We assume that the domain $\mathcal{D}$ of value of each variable is the set of integers between two bounds min and max. These bounds are

chosen such that min<0<max.

The verification conditions (after trivial simplifications) are the following (where $x,y,q \in [min,max]$) :

$P_6(x,y,q)$

$P_6(x,y,q) \Rightarrow [(x<y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$
$P_5(x,y,q) \Rightarrow P_4(x+y,y,q)$
$P_4(x,y,q) \Rightarrow P_3(x,y,q-1)$
$P_3(x,y,q) \Rightarrow [(x \geq y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$
$P_2(x,y,q) \Rightarrow [\forall q' \in [min,max], (q=0) \Rightarrow P_1(x,y,q')]$

$[(0 \leq x \leq max) \wedge (y=0)] \Rightarrow \neg P_1(x,y,q)$

Intuitively, by reductio ad absurdum, if y=0 the program could terminate only if x<0, in contradiction with the hypothesis on the initial value of x. This can be proved formally using the following invariants :

$P_i(x,y,q) = [(y=0) \Rightarrow (x<0)]$ for i=1,...,5
$P_6(x,y,t) = tt$

9.7 COMPLETENESS CHECK

According to paragraph 7.5 we have to check that :

$\forall I \in A[\![P]\!],\ VC[\![P]\!](\varepsilon,\sigma)(\psi)(I) \Rightarrow \widetilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\widetilde{\rho}[\![P]\!](I))$

Using the fact that $\widetilde{VC}[\![P]\!](\varepsilon,\sigma)(\psi)(\widetilde{I}) = VC[\![P]\!](\varepsilon,\sigma)(\psi)(\rho[\![P]\!](\widetilde{I}))$ it is sufficient to check that $\rho[\![P]\!](\widetilde{\rho}[\![P]\!](I)) = I$, which is trivial.

In fact, $\rho[\![P]\!]$ is a bijection between $\widetilde{A}[\![P]\!]$ and $A[\![P]\!]$ (and $\widetilde{\rho}[\![P]\!]$ is its inverse). $\widetilde{VC}[\![P]\!]$ could have been chosen weak enough not to be complete. Such a case of incompleteness is not fundamental since $\widetilde{VC}[\![P]\!]$ could be strengthened enough in order to be equivalent to $VC[\![P]\!]$. Then, as above, the only difference between $VC[\![P]\!]$ and $\widetilde{VC}[\![P]\!]$ is that the invariants I and $\widetilde{I}$ are dissimilar but isomorphic descriptions of sets of states.

In general $A[\![P]\!]$ and $\widetilde{A}[\![P]\!]$ are not isomorphic and $\widetilde{A}[\![P]\!]$ may not be expressive enough in order to describe all possible sets of states during execution (e.g. Wand[78]). This choice can lead to a fundamental incompleteness result, in the sense that for that choice of $\widetilde{A}[\![P]\!]$, no complete verification condition $\widetilde{VC}[\![P]\!]$ might exist (e.g. Clarke[77]).

10. EXAMPLE : DESIGN OF A METHOD FOR PROVING THE CLEAN BEHAVIOR OF PROGRAMS

Let $P \in \mathcal{P}$ be a program, and $\phi \in (M[\![P]\!] \to \{tt, ff\})$ be a characterization of the possible initial values of the program variables. Execution of program P does not lead to a run-time error if and only if any reachable state during execution which is not an exit state has a possible successor state, that is

$$[\forall \underline{s}, \overline{s} \in S[\![P]\!],\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s}, \overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$$

where

$$\varepsilon(\langle L, m\rangle) = [(P \Xi L{:}\alpha) \wedge \phi(m)]$$
$$\sigma(\langle L, m\rangle) = [\neg(P \Xi \alpha L{:})]$$
$$\psi(\langle L, m\rangle) = [\exists L' \in C[\![P]\!], m' \in M[\![P]\!] \mid t[\![P]\!](\langle L, m\rangle, \langle L', m'\rangle)]$$

Notice that the only difference with paragraph 9.1 is σ and ψ. Therefore choosing the induction principle and assertion language considered at paragraph 9, the verification conditions will be those of 9.4, except for the basis :

$$\begin{aligned}
&\widetilde{VC}_\sigma[\![P]\!](\psi, \sigma)(\widetilde{I}) \\
&\quad = VC_\sigma[\![P]\!](\psi, \sigma)(\rho[\![P]\!](\widetilde{I})) \\
&\quad = [\forall \overline{s} \in S[\![P]\!], [\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \rho[\![P]\!](\widetilde{I})(\overline{s}) \\
&\quad = [\forall L \in C[\![P]\!], m \in M[\![P]\!], [\neg(P \Xi \alpha L{:}) \wedge (\forall L' \in C[\![P]\!], m' \in M[\![P]\!], \\
&\qquad\qquad \neg\, t[\![P]\!](\langle L, m\rangle, \langle L', m'\rangle)] \Rightarrow \widetilde{I}(L)(m)
\end{aligned}$$

Therefore for all labels L of program P but the exit one, we must prove :

$$[\forall L' \in C[\![P]\!], m' \in M[\![P]\!], \neg\, t[\![P]\!](\langle L, m\rangle, \langle L', m'\rangle)] \Rightarrow \widetilde{I}(L)(m)$$

This condition can be further refined according to the various cases corresponding to the definition of $t[\![P]\!]$:

(a) - If $P \Xi \alpha L{:}\underline{skip};L'{:}\beta$ or $P \Xi \alpha L{:}\underline{else}\ P'\ \underline{fi};L'{:}\beta$ or $P \Xi \alpha L{:}\underline{fi};L'{:}\beta$ then the left hand side is false and the verification condition is identically true.

(b) - $P \Xi \alpha L{:}v{:=}E;L'{:}\beta$

$$[m \notin dom(\mathbb{E}[\![E]\!])] \Rightarrow \widetilde{I}(L)(m)$$

(c) - $P \Xi \alpha L{:}\underline{if}\ B\ \underline{then}\ \beta$ or $P \Xi \alpha L{:}\underline{while}\ B\ \underline{do}\ \beta$ or $P \Xi \alpha \underline{while}\ B\ \underline{do}\ L_1{:}C_1;\ldots;L_n{:}C_n;L{:}\underline{od};\beta$

$$[m \notin dom(\mathbb{B}[\![B]\!])] \Rightarrow \widetilde{I}(L)(m)$$

We can recapitulate the verification conditions as follows :

- Null command :

$$\ldots L_1{:}\underline{skip};L_2{:}\ldots \qquad\qquad P_2 \Rightarrow P_1$$

- Assignment command :

$...L_1: v:=E; L_2:...$ $\quad \neg dom(E) \Rightarrow P_1$, $P_2 \Rightarrow [\forall v', (v=E|^{v'}_{v}) \Rightarrow P_1|^{v'}_{v}]$

- Conditional command :

$...L_1:$ if B then $L_2:... L_3:...$ else $L_4:... L_5:...$ fi; $L_6:...$

$\neg dom(B) \Rightarrow P_1$, $P_2 \Rightarrow [B \Rightarrow P_1]$, $P_4 \Rightarrow [\neg B \Rightarrow P_1]$, $P_6 \Rightarrow [P_3 \wedge P_5]$

- Iteration command :

$...L_1:$ while B do $L_2:... L_3:...$ od; $L_4:...$

$\neg dom(B) \Rightarrow P_1$, $P_2 \Rightarrow [B \Rightarrow (P_1 \wedge P_3)]$, $P_4 \Rightarrow [\neg B \Rightarrow (P_1 \wedge P_3)]$

- Entry point :

$L_1:...$ $\quad \phi \Rightarrow \neg P_1$

For our example program :

```
1: Q:=0;
2: while X≥Y do
3:     Q:=Q+1;
4:     X:=X-Y;
5: od;
6:
```

we can prove that no run-time error is possible whenever the initial values $\underline{x},\underline{y}$ of X and Y are such that $\underline{x}\geq 0 \wedge \underline{y}>0$ and $\mathcal{D}$=[min,max].

The verification conditions (after trivial simplifications) are :

$[(q+1)>max] \Rightarrow P_3(x,y,q)$

$[((x-y)<min) \vee ((x-y)>max)] \Rightarrow P_4(x,y,q)$

$P_6(x,y,q) \Rightarrow [(x<y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$

$P_5(x,y,q) \Rightarrow P_4(x+y,y,q)$

$P_4(x,y,q) \Rightarrow P_3(x,y,q-1)$

$P_3(x,y,q) \Rightarrow [(x\geq y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$

$P_2(x,y,q) \Rightarrow [\forall q' \in [min,max], (q=0) \Rightarrow P_1(x,y,q')]$

$[x\geq 0 \wedge y>0] \Rightarrow \neg P_1(x,y,q)$

Intuitively, if initially $\underline{x}\geq 0$ and $\underline{y}>0$ the only possible run-time error is an overflow in statement 3:Q:=Q+1;. Since Q remains positive, this can happen

only if the initial value $\underline{x}$ of X (that is q*y+x in term of the current values of variables) is greater than max, in contradiction with the fact that $\underline{x} \in \mathcal{D} = [min,max]$.

This reasoning is formalized by showing that the following assertions satisfy the above verification conditions (where $x,y,q \in [min,max]$) :

$P_1(x,y,q) = [\neg(x \geq 0 \wedge y > 0)]$
$P_2(x,y,q) = P_5(x,y,q) = [(x \geq 0 \wedge y > 0 \wedge q \geq 0) \Rightarrow (q*y+x > max)]$
$P_3(x,y,q) = [(x \geq y > 0 \wedge q \geq 0) \Rightarrow (q*y+x > max)]$
$P_4(x,y,q) = [(x \geq y > 0 \wedge q \geq 1) \Rightarrow ((q-1)*y+x > max)]$
$P_6(x,y,q) = ff$

11. EXAMPLE : DESIGN OF A METHOD FOR PROVING GLOBAL INVARIANTS OF PROGRAMS

Let $P \in \mathcal{P}$ be a program. A global invariant of program P is an assertion $\delta \in (M[\![P]\!] \rightarrow \{tt,ff\})$ on the values of the variables which is always true during program execution. This means that :

$$[\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi(\overline{s})]$$

where

$\varepsilon(<L,m>) = [(P \equiv L:\alpha) \wedge \phi(m)]$
$\sigma(<L,m>) = tt$
$\psi(<L,m>) = \delta(m)$

Choosing the induction principle and assertion language considered at paragraphs 9.1 and 10, the verification conditions will be those of 9.4, except for the basis :

$$\begin{aligned}
&\widetilde{VC}_\sigma[\![P]\!](\psi,\sigma)(\widetilde{I}) \\
&\quad = VC_\sigma[\![P]\!](\psi,\sigma)(\rho[\![P]\!](\widetilde{I})) \\
&\quad = [\forall \overline{s} \in S[\![P]\!], [\neg\psi(\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \rho[\![P]\!](\widetilde{I})(\overline{s})] \\
&\quad = [\forall L \in C[\![P]\!], m \in M[\![P]\!], \neg\delta(m) \Rightarrow \widetilde{I}(L)(m)]
\end{aligned}$$

so that for all program labels L we have to prove that :

$$\neg\delta(m) \Rightarrow \widetilde{I}(L)(m)$$

For our example program :

```
1: Q:=0;
2: while X≥Y do
3:     Q:=Q+1;
4:     X:=X-Y;
5: od;
6:
```

the verification conditions are the following (where $x,y,q \in [min,max]$) :

$$\neg\delta(x,y,q) \Rightarrow P_i(x,y,q) \qquad i=1,2,\ldots,6$$

$$P_6(x,y,q) \Rightarrow [(x<y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$$
$$P_5(x,y,q) \Rightarrow P_4(x+y,y,q)$$
$$P_4(x,y,q) \Rightarrow P_5(x,y,q-1)$$
$$P_3(x,y,q) \Rightarrow [(x \geq y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$$
$$P_2(x,y,q) \Rightarrow [\forall q' \in [min,max], (q=0) \Rightarrow P_1(x,y,q')]$$

$$\phi(x,y,q) \Rightarrow \neg P_1(x,y,q)$$

In order to prove that $\phi(x,y,q)=(x \geq 0 \wedge y>0 \wedge q \geq 0)$ is a global invariant of this program we can choose the following assertions :

$$P_1(x,y,q)=P_2(x,y,q)=P_5(x,y,q)=[x<0 \vee y \leq 0 \vee q<0]$$
$$P_3(x,y,q)=P_4(x,y,q)=[(x \geq y) \Rightarrow (x<0 \vee y \leq 0 \vee q<0)]$$
$$P_6(x,y,q)=[(x<y) \Rightarrow (x<0 \vee y \leq 0 \vee q<0)]$$

12. CONCLUSION : CHOOSING COMPLEMENTARY METHODS FOR PROVING DIFFERENT INVARIANCE PROPERTIES OF PROGRAMS

If different invariance properties have to be proved about a program (e.g. global invariance, clean behavior and partial correctness) it is usually recommended to prove these properties in turn in order to separate concerns. For example, in order to prove :

$$\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow [\psi_1(\overline{s}) \wedge \psi_2(\overline{s}) \wedge \psi_3(\underline{s},\overline{s})]$$

where

$$\varepsilon(\langle \underline{L},\underline{m}\rangle)=[(P \equiv \underline{L}:\alpha) \wedge \phi(\underline{m})] \quad \text{(characterizes entry states)}$$
$$\sigma(\langle \overline{L},\overline{m}\rangle)=tt$$
$$\psi_1(\langle \overline{L},\overline{m}\rangle)=\delta(\overline{m}) \quad \{\delta \text{ is globally invariant}\}$$
$$\psi_2(\langle \overline{L},\overline{m}\rangle)=[\neg(P \equiv \alpha\overline{L}:) \Rightarrow (\exists L' \in C[\![P]\!],\ m' \in M[\![P]\!] \mid t[\![P]\!](\langle \overline{L},\overline{m}\rangle,\langle L',m'\rangle)]$$
(clean behavior)
$$\psi_3(\langle \underline{L},\underline{m}\rangle,\langle \overline{L},\overline{m}\rangle)=[(P \equiv \alpha\overline{L}:) \Rightarrow \psi(\underline{m},\overline{m})] \quad \text{(partial correctness)}$$

we can prove successively that :

$$\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi_1(\overline{s})$$
$$\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi_2(\overline{s})$$
$$\forall \underline{s},\overline{s} \in S[\![P]\!], [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi_3(\underline{s},\overline{s})$$

Since the proofs are independent, we can use for each of these properties a different invariance proof method, that is basically, a different induction principle. However, although the proofs have been separated, it is better to use proof methods which avoid duplications, that is such that :

- The verification conditions for all proofs will be the same (except for those conditions which depend upon the particular property ψ_i which is proved invariant),
- A common inductive invariant will exist for all proofs.

In the case of global invariance, clean behavior and partial correctness, these requirements are fulfilled by Floyd-Naur-Hoare's method and in our opinion this partly explains its success.

If induction principle (I) (or $(\tilde{I})$) is used for all three proofs, the verification conditions $(I.\varepsilon)$ and $(I.i)$ are common to all proofs and only $(I.\sigma)$ depends upon the particular property ψ_i which is proved invariant. Moreover, there exists common inductive invariants for all proofs, namely $I(\underline{s},s)=[\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},s)]$ is the strongest one.

These requirements are not fulfilled by Morris & Wegbreit's subgoal induction method. At first, this is surprising, since the above argument which was used for (I) can be repeated for (I^{-1}). However the strongest common invariant for all proofs is now $I(s,\overline{s})=[t[\![P]\!]^*(s,\overline{s}) \wedge \sigma(\overline{s})]$. Since $\sigma(\overline{s})=tt$, this invariant characterizes all possible ascendants s of all possible states $\overline{s}$ of the program! (Whereas for Floyd-Naur-Hoare's method, the invariant characterizes all possible descendants s of only the entry states $\underline{s}$ of the program). Since this is often untractable, several remedies can be proposed, but none of them is without defects :

- Following Morris & Wegbreit[77], we can choose a smaller set of final states. For example, for partial correctness, we can prove :

$$\forall \underline{s},\overline{s} \in S[\![P]\!],\ [\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},\overline{s}) \wedge \sigma(\overline{s})] \Rightarrow \psi_3(\underline{s},\overline{s})$$

where

$$\varepsilon(\langle \underline{L},m\rangle) = [(P \equiv \underline{L}:\alpha) \wedge \phi(m)]$$
$$\sigma(\langle \overline{L}\ \overline{m}\rangle) = [P \equiv \alpha \overline{L}:]$$
$$\psi_3(\langle \underline{L},\underline{m}\rangle,\langle \overline{L},\overline{m}\rangle) = \psi(\underline{m},\overline{m})$$

The defect is now that, in general, the invariant can no longer be used for proving global invariance and clean behavior.

- An alternative consists in using a more precise invariant $I(s,\overline{s})$ which, instead of characterizing all possible ascendants s of possible states $\overline{s}$ of the program (i.e. $I(s,\overline{s})=[t[\![P]\!]^*(s,\overline{s}) \wedge \sigma(\overline{s})]$ where $\sigma(\overline{s})=tt$), will characterize only those states s which are also possible descendants of the entry states. The strongest invariant with this property is :

$$I(s,\overline{s}) = [[\exists \underline{s} \in S[\![P]\!] \mid \varepsilon(\underline{s}) \wedge t^*[\![P]\!](\underline{s},s)] \Rightarrow [\ t^*[\![P]\!](s,\overline{s}) \wedge \sigma(\overline{s})]]$$

The defect is now that the inductive invariant contains as a subpart the invariant of Floyd-Naur-Hoare's method in the shape of an hypothesis.

- The above two ideas can be combined. On one hand we can use an invariant $I(\underline{s},s)$ à la Floyd-Naur-Hoare which satisfies $(I.\varepsilon)$ and $(I.i)$. Then $[\varepsilon(\underline{s}) \wedge t[\![P]\!]^*(\underline{s},s)] \Rightarrow I(\underline{s},s)$, $I(\underline{s},s)$ describes what has happened up to now and

can be used for proving global invariance and clean behavior. On the other hand, for partial correctness we can use another invariant $J(s,\overline{s})$, which describes what remains to be done until the end of the program and such that $[[\exists \underline{s} \in S[\![P]\!] \mid I(\underline{s},s)] \Rightarrow J(s,\overline{s})]$ satisfies $(I^{-1}.\sigma)$ and $(I^{-1}.i)$ where $\sigma(<\overline{L},\overline{m}>) = [P \equiv \alpha\overline{L}:]$.
The defect is now that we have a redundant description of what the program does (i.e. at each program point we describe the past (by $I(\underline{s},s)$) and the future (by $J(s,\overline{s})$)). Also, the proof is given twice. However, this defect can turn out to be an advantage since readers of the program will get a better description of its behavior while redundant but different proofs can be useful to avoid errors.

We illustrate this last approach on our example program. At each program point i, we have interleaved a comment $\{P_i(\underline{x},\underline{y},x,y,q,\overline{x},\overline{y},\overline{q})\}$ which is of the form $\{[B_i(\underline{x},\underline{y},x,y,q)] \rightsquigarrow [A_i(x,y,q,\overline{x},\overline{y},\overline{q})]\}$. B_i describes the relationship between the entry and current values of the variables, while A_i describes the relationship between the current and exit values of the variables.

1: $\{[x=\underline{x} \wedge y=\underline{y} \wedge \underline{x}\geq 0 \wedge \underline{y}>0] \rightsquigarrow [x=\overline{q}*\overline{y}+\overline{x} \wedge y=\overline{y} \wedge 0\leq\overline{x}<\overline{y}]\}$
Q:=0;
2: $\{[x=\underline{x} \wedge y=\underline{y} \wedge \underline{x}\geq 0 \wedge \underline{y}>0 \wedge q=0] \rightsquigarrow [x=\overline{q}*\overline{y}+\overline{x} \wedge y=\overline{y} \wedge 0\leq\overline{x}<\overline{y}]\}$
while X≥Y **do**
3: $\{[\underline{x}=q*y+x \wedge y=\underline{y} \wedge \underline{y}>0 \wedge x\geq y \wedge q\geq 0] \rightsquigarrow [x=(\overline{q}-q)*y+\overline{x} \wedge y=\overline{y} \wedge 0\leq\overline{x}<\overline{y}]\}$
Q:=Q+1;
4: $\{[\underline{x}=(q-1)*y+x \wedge y=\underline{y} \wedge \underline{y}>0 \wedge x\geq y \wedge q\geq 1] \rightsquigarrow [x=(\overline{q}-q+1)*y+\overline{x} \wedge y=\overline{y} \wedge 0\leq\overline{x}<\overline{y}]\}$
X:=X-Y;
5: $\{[\underline{x}=q*y+x \wedge y=\underline{y} \wedge \underline{y}>0 \wedge x\geq 0 \wedge q\geq 1] \rightsquigarrow [x=(\overline{q}-q)*y+\overline{x} \wedge y=\overline{y} \wedge 0\leq\overline{x}<\overline{y}]\}$
od;
6: $\{[\underline{x}=q*y+x \wedge y=\underline{y} \wedge \underline{x}\geq 0 \wedge \underline{y}>0 \wedge 0\leq x<y \wedge q\geq 0] \rightsquigarrow [x=\overline{x} \wedge y=\overline{y} \wedge q=\overline{q}]\}$

The comments are clearly redundant as far as the proof is concerned. However, in our opinion, this redundancy is useful to the program reader, who given only B_i or A_i, has to guess the other one in order to understand the program.

The verification conditions are the following :

- $B_i(\underline{x},\underline{y},x,y,q)$ satisfies :

$I_1(\underline{x},\underline{y},x,y,q) \Leftarrow [x=\underline{x} \wedge y=\underline{y} \wedge \underline{x}\geq 0 \wedge \underline{y}>0]$
$I_2(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q' \in [min,max] \mid I_1(\underline{x},\underline{y},x,y,q') \wedge q=0]$
$I_3(\underline{x},\underline{y},x,y,q) \Leftarrow [(I_2(\underline{x},\underline{y},x,y,q) \vee I_5(\underline{x},\underline{y},x,y,q)) \wedge x\geq y]$

$I_4(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q' \in [min,max] | I_3(\underline{x},\underline{y},x,y,q') \wedge q=q'+1]$
$I_5(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists x' \in [min,max] | I_4(\underline{x},\underline{y},x',y,q) \wedge x=x'-y]$
$I_6(\underline{x},\underline{y},x,y,q) \Leftarrow [(I_2(\underline{x},\underline{y},x,y,q) \vee I_5(\underline{x},\underline{y},x,y,q)) \wedge x<y]$

- $[\exists \underline{x},\underline{y} \in [min,max] | B_i(\underline{x},\underline{y},x,y,q)] \Rightarrow A_i(x,y,q,\overline{x},\overline{y},\overline{q})$ satisfies :

$J_1(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow J_2(x,y,0,\overline{x},\overline{y},\overline{q})$
$J_2(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow [(J_3(x,y,q,\overline{x},\overline{y},\overline{q}) \wedge x \geq y) \vee (J_6(x,y,q,\overline{x},\overline{y},\overline{q}) \wedge x<y)]$
$J_3(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow [(q<max) \wedge J_4(x,y,q+1,\overline{x},\overline{y},\overline{q})]$
$J_4(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow [(min \leq x-y \leq max) \wedge J_5(x-y,y,q,\overline{x},\overline{y},\overline{q})]$
$J_5(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow [(J_3(x,y,q,\overline{x},\overline{y},\overline{q}) \wedge x \geq y) \vee (J_6(x,y,q,\overline{x},\overline{y},\overline{q}) \wedge x<y)]$
$J_6(x,y,q,\overline{x},\overline{y},\overline{q}) \Leftarrow [x=\overline{x} \wedge y=\overline{y} \wedge q=\overline{q}]$

(Notice that it is sufficient to prove that the above verification conditions are satisfied by $A_i(x,y,q,\overline{x},\overline{y},\overline{q})$. However, in general, an invariant of the form $B_i \Rightarrow A_i$ is more natural since if the input satisfies certain constraints B_i then the output is to have certain properties specified by A_i, whereas if the input contraints B_i are violated we can leave the output unspecified because such inputs can never occur in program operation).

- For proving that $(x \geq 0 \wedge y>0 \wedge q \geq 0)$ is invariant at program points 2 to 6, we must verify that :

$B_i(\underline{x},\underline{y},x,y,q) \Rightarrow (x \geq 0 \wedge y>0 \wedge q \geq 0)$ for i=2,...,6

- For proving the absence of run-time errors, we must verify that :

$B_3(\underline{x},\underline{y},x,y,q) \Rightarrow (q<max)$
$B_4(\underline{x},\underline{y},x,y,q) \Rightarrow (min \leq x-y \leq max)$

- For proving partial correctness we must prove that :

$B_6(\underline{x},\underline{y},x,y,q) \Rightarrow \psi(\underline{x},\underline{y},x,y,q)$

or

$A_1(x,y,q,\overline{x},\overline{y},\overline{q}) \Rightarrow \psi(x,y,\overline{x},\overline{y},\overline{q})$

where

$\psi(\underline{x},\underline{y},\overline{x},\overline{y},\overline{q}) = [(\underline{x} \geq 0 \wedge \underline{y}>0) \Rightarrow (\underline{x}=\overline{q}*\overline{y}+\overline{x} \wedge \underline{y}=\overline{y} \wedge 0 \leq \overline{x}<\overline{y})]$

13. REFERENCES

Apt K.R.[1981], *Ten years of Hoare's logic : A survey*, ACM-TOPLAS, Vol.3, Nb.4, (Oct.1981), 431-483.

Clarke E.M.[1977], *Programming language constructs for which it is impossible to obtain "good" Hoare-like axiom systems*, Conf. Rec. 4th ACM Symp. on Principles of Prog. Lang., (Jan.1977), 10-20.

Cousot P.[1979], *Analysis of the behavior of dynamic discrete systems*, Research Report 161, IMAG, Grenoble University, France, (Jan.1979).

Cousot P.[1981], *Semantic foundations of program analysis*, in "Program Flow Analysis : Theory and Applications", Muchnick S.S. & Jones N.D. (Eds.), Ch.10 Prentice Hall Inc., (1981), 303-342.

Cousot R.[1981], *Proving invariance properties of parallel programs by backward induction*, Research Report, CRIN-81-P026, Nancy, France, (March 1981).

Cousot P. & Cousot R.[1980a], *Semantic analysis of communicating sequential processes*, Automata, Languages and Programming, 7th Colloq., Noordwijkerhout Lecture Notes in Comp. Sci. 85, Springer-Verlag, (July 1980), 119-133.

Cousot P. & Cousot R.[1980b], *Reasoning about invariance proof methods*, Proc. Int. Workshop on Program Construction, Château de Bonas, France, Tome 1, INRIA ed., (Sept. 1980).

Floyd R.W.[1967], *Assigning meaning to programs*, Proc. Symp. in Applied Math. Vol.19, AMS, Providence, R.I., (1967), 19-32.

Hoare C.A.R.[1969], *An axiomatic basis for computer programming*, CACM 12, 10(Oct. 1969), 576-580,583.

King J.C.[1979], *Program correctness : on inductive assertion methods*, RJ2525, Comp. Sci. Dept., IBM Research, San Jose,CA, (Aug.1979), 44p.

Manna Z.[1971], *Mathematical theory of partial correctness*, J.Comp. Syst. Sci. 5, 3(June 1971), 239-253.

Naur P.[1966], *Proof of algorithms by general snapshots*, BIT 6, (1966), 310-316.

Morris J.H. & Wegbreit B.[1977], *Subgoal induction*, CACM 20, 4(April 1977), 209-222.

Wand M.[1978], *A new incompleteness result for Hoare's system*, JACM 25, 1(Jan. 1978), 168-175.