Conference Record
of the
SIXTH ANNUAL ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES

Papers Presented at the Symposium
San Antonio, Texas
January 29-31, 1979

Sponsored by the
ASSOCIATION FOR COMPUTING MACHINERY
SPECIAL INTEREST GROUP ON AUTOMATA AND COMPUTABILITY THEORY
SPECIAL INTEREST GROUP ON PROGRAMMING LANGUAGES

SYSTEMATIC DESIGN OF PROGRAM ANALYSIS FRAMEWORKS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique,U.S.M.G., BP.53X
38041 Grenoble cedex, France

1. INTRODUCTION and SUMMARY

Semantic analysis of programs is essential in optimizing compilers and program verification systems. It encompasses data flow analysis, data type determination, generation of approximate invariant assertions, etc.

Several recent papers (among others Cousot & Cousot[77a], Graham & Wegman[76], Kam & Ullman[76], Kildall[73], Rosen[78], Tarjan[76], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a *program analysis framework* (A,t,γ) where A is a lattice of (approximate) assertions, t is an (approximate) predicate transformer and γ is an often implicit function specifying the meaning of the elements of A. This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics.

Preliminary definitions are given in Section 2 concerning the merge over all paths and (least) fixpoint program-wide analysis methods. In Section 3 we briefly define the (forward and backward) deductive semantics of programs which is later used as a formal basis in order to prove the correctness of the approximate program analysis frameworks. Section 4 very shortly recall the main elements of the lattice theoretic approach to approximate semantic analysis of programs.

The design of a space of approximate assertions A is studied in Section 5. We first justify the very reasonable assumption that A must be chosen such that the exact invariant assertions of any program must have an upper approximation in A and that the approximate analysis of any program must be performed using a deterministic process. These assumptions are shown to imply that A is a Moore family, that the approximation operator (wich defines the least upper approximation of any assertion) is an upper closure operator and that A is necessarily a complete lattice. We next show that the connection between a space of approximate assertions and a computer representation is naturally made using a pair of isotone adjoined functions. This type of connection between two complete lattices is related to Galois connections thus making available classical mathematical results. Additional results are proved, they hold when no two approximate assertions have the same meaning.

This work was supported in part by (*) CNRS, Laboratoire Associé n°7 and ATP-Informatique D3119 and in part by (**) IRIA-SESORI.

In Section 6 we study and examplify various methods which can be used in order to define a space of approximate assertions or equivalently an approximation function. They include the characterization of the least Moore family containing an arbitrary set of assertions, the construction of the least closure operator greater than or equal to an arbitrary approximation function, the definition of closure operators by composition, the definition of a space of approximate assertions by means of a complete join congruence relation or by means of a family of principal ideals.

Section 7 is dedicated to the design of the approximate predicate transformer induced by a space of approximate assertions. First we look for a reasonable definition of the correctness of approximate predicate transformers and show that a local correctness condition can be given which has to be verified for every type of elementary statement. This local correctness condition ensures that the (merge over all paths or fixpoint) global analysis of any program is correct. Since isotony is not required for approximate predicate transformers to be correct it is shown that non-isotone program analysis frameworks are manageable although it is later argued that the isotony hypothesis is natural. We next show that among all possible approximate predicate transformers which can be used with a given space of approximate assertions there exists a best one which provides the maximum information relative to a program-wide analysis method. The best approximate predicate transformer induced by a space of approximate assertions turns out to be isotone. Some interesting consequences of the existence of a best predicate transformer are examined. One is that we have in hand a formal specification of the programs which have to be written in order to implement a program analysis framework once a representation of the space of approximate assertions has been chosen. Examples are given, including ones where the semantics of programs is formalized using Hoare[78]'s sets of traces.

In Section 8 we show that a hierarchy of approximate analyses can be defined according to the fineness of the approximations specified by a program analysis framework. Some elements of the hierarchy are shortly exhibited and related to the relevant literature.

In Section 9 we consider global program analysis methods. The distinction between "distributive" and "non-distributive" program analysis frameworks is studied. It is shown that when the best approximate predicate transformer is considered the coincidence or not of the merge over all paths and least fixpoint global analyses of programs is a consequence of the choice of the space of approximate assertions. It is shown that the space of approximate assertions can always be refined so that the merge over all paths analysis of a program can be defined by means of a least fixpoint of isotone equations.

Section 10 is devoted to the combination of program analysis frameworks. We study and examplify how to perform the "sum", "product" and "power" of program analysis frameworks. It is shown that combined analyses lead to more accurate information than the conjunction of the corresponding separate analyses but this can only be achieved by a new design of the approximate predicate transformer induced by the combined program analysis frameworks.

2. PRELIMINARY DEFINITIONS

A *program* π is a pair (V,G) where G is a program graph and V is the universe in which the program variables take their values.

The set L of *elementary commands* consists in elementary tests and elementary assignments : $L = L_t \cup L_a$. An *elementary test* $q \in L_t$ is a total map from $dom(q) \subseteq V$ into $B = \{true, false\}$. An *elementary assignment* $e \in L_a$ is a total map from $dom(e) \subseteq V$ into V.

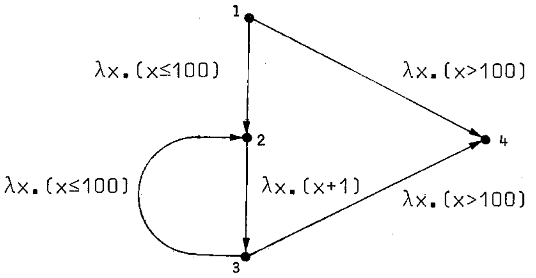
A *program graph* G is a tuple (n, E, n_i, n_0, C) where n is the number of vertices (therefore $n \geq 1$), $E \subseteq [1,n]^2$ is the (non-empty) set of edges, $n_i \in [1,n]$ is the entry point, $n_0 \in [1,n]$ is the exit point and $C \in (E \to L)$ defines the command $C(<i,j>)$ associated with each $<i,j>$ in E. Let $pred \in [1,n] \to 2^{[1,n]}$ be $\lambda j.\{i \in [1,n] : <i,j> \in E\}$ and $succ \in [1,n] \to 2^{[1,n]}$ be $\lambda i.\{j \in [1,n] : <i,j> \in E\}$, then we assume that $pred(n_i) = \emptyset$, $succ(n_0) = \emptyset$ and for any $v \in [1,n] - \{n_i, n_0\}$, $pred(v) \neq \emptyset$ and $succ(v) \neq \emptyset$.

Example 2.0.1

The program :

```
      begin
{1}      while x≤100 do        {x is an integer variable}
{2}         x := x+1;          {no overflow can occur}
{3}      od;
{4}   end
```

will be represented by its program graph :



End of example.

If $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ is a complete lattice, $t \in (L \to (A \to A))$ and $\phi \in A$ then the *merge over all paths analysis of* π *using* (A,t) *and* ϕ $(MOP_\pi(t,\phi))$ is $P \in A^n$ defined as :

$$\forall i \in [1,n],\ P_i = \bigsqcup_{p \in path(i)} \tilde{t}(p)(\phi)$$

where $path(i)$ is the set of paths from the entry point n_i to the vertex i and $\tilde{t} \in (E^* \to (A \to A))$ is recursively defined as follows : if p is an empty path then $\tilde{t}(p)$ is the identity map on A else $p = (q,a)$ where $q \in E^*$, $a \in E$ and $\tilde{t}(p) = \lambda\phi.t[C(a)][\tilde{t}(q)(\phi)]$.

The *system of equations* $P = F_\pi(t,\phi)(P)$ *associated with the program* π *using* (A,t) *and* ϕ is defined as follows :

$$\begin{cases} P_{n_i} = \phi \\ P_j = \bigsqcup_{i \in pred(j)} t(C(<i,j>))(P_i) & \text{if } j \in [1,n] - \{n_i\} \end{cases}$$

Notation : If $M(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ is a complete lattice then the set $(L \to M)$ of total maps from the set L into M is a complete lattice $(L \to M)(\sqsubseteq', \bot', \top', \sqcup', \sqcap')$ for the pointwise ordering $f \sqsubseteq' g$ iff $\forall x \in L,\ f(x) \sqsubseteq g(x)$. In the following the distinction between $\sqsubseteq$, $\bot$, $\top$, $\sqcup$, $\sqcap$ and $\sqsubseteq'$, $\bot'$, $\top'$, $\sqcup'$, $\sqcap'$ will be determined by the context. Also a map $f \in (L \to M)$ will be extended to $(2^L \to 2^M)$ as $\lambda S \in 2^L.\{f(x) : x \in S\}$ and to $(L^n \to M^n)$ as $\lambda<x_1,\ldots,x_n>.(<f(x_1),\ldots,f(x_n)>)$.

3. DEDUCTIVE SEMANTICS OF PROGRAMS

3.1 Forward Semantics

The *forward semantic analysis* of a program π consists in determining at each program point an invariant assertion which characterizes the set of states which are the descendants of the input states satisfying a given entry assertion ϕ.

More precisely an *assertion* is a total map from V into B. The set $A = (V \to B)(\Rightarrow, \lambda X \in V.false, \lambda X \in V.true, \vee, \wedge, \neg)$ of assertions is a complete boolean lattice partially ordered by the implication $\Rightarrow$.

Let $sp(S)(P)$ be Floyd[67]'s strongest post-condition derived from the pre-condition $P \in A$ for the elementary command $S \in L$. We assume that the operational semantics of the elementary commands is such that for an elementary test we have :

$$sp(q) = \lambda P \in A.[\lambda X \in V.(P(X) \wedge X \in dom(q) \wedge q(X))]$$

whereas for an elementary assignment e we have :

$$sp(e) = \lambda P \in A.[\lambda X \in V.(\exists Y \in V : P(Y) \wedge Y \in dom(e) \wedge X = e(Y))]$$

(Notice that for all $S \in L$, $sp(S)$ is a complete join-morphism (i.e. $\forall A \subseteq A$, $sp(S)(\vee A) = \vee sp(S)(A)$)).

We assume that the operational semantics of the program π is such that at each program point $i \in [1,n]$ the invariant assertion P_i which characterizes the set of states which are the descendants of the input states satisfying a given entry assertion $\phi \in A$ is the merge over all paths analysis of π using sp and ϕ. P is the least fixpoint $lfp(F_\pi(sp,\phi))$ of the system of equations $P = F_\pi(sp,\phi)(P)$ associated with the program π using sp and ϕ.

Example 3.1.0.1

The system of forward semantic equations associated with the program 2.0.1 is :

$$\begin{cases} P_1 = \phi \\ P_2 = sp(\lambda x.[x\leq 100])(P_1 \vee P_3) \\ P_3 = sp(\lambda x.[x+1])(P_2) \\ P_4 = sp(\lambda x.[x>100])(P_1 \vee P_3) \end{cases}$$

taking $\phi=\lambda x.(x=1)$ its least fixpoint characterizes the descendants of the input states satisfying ϕ :

$$\left[\begin{array}{l} P_1 = \lambda x.(x=1) \\ P_2 = \lambda x.(1\leq x\leq 100) \\ P_3 = \lambda x.(2\leq x\leq 101) \\ P_4 = \lambda x.(x=101) \end{array}\right.$$

End of Example.

3.2 Backward Semantics

The backward semantic analysis of a program consists in determining at each program point an invariant assertion which characterizes the set of states which are the ascendants of the output states satisfying a given exit specification ϕ.

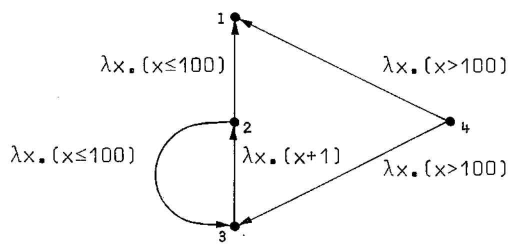
Since we can consider the inverse of the state transition relation defined by the operational semantics no new formalism is necessary in order to treat backward program analysis. Instead of Floyd's forward predicate transformer we just have to consider Hoare[69]-Dijkstra[76]'s backward predicate transformer :

$$wp(q) = \lambda P\epsilon A.[\lambda X\epsilon V.(P(X) \wedge X\epsilon dom(q) \wedge q(X))]$$
$$wp(e) = \lambda P\epsilon A.[\lambda X\epsilon V.(X\epsilon dom(e) \wedge P(e(X)))]$$

(notice that $\forall S\epsilon L$, $wp(S)$ is a complete join and meet morphism) and the *inverted program graph* $G'=(n,E',n_0,n_i,C')$ where $E'=\{<i,j> : <j,i>\epsilon E\}$, $C'=\lambda<i,j>\epsilon E'.[C(<j,i>)]$.

Example 3.2.0.1

The inverted program graph corresponding to 2.0.1 is :



The corresponding system of backward semantic equations is :

$$\begin{cases} P_1 = wp(\lambda x.[x\leq 100])(P_2) \vee wp(\lambda x.[x>100])(P_4) \\ P_2 = wp(\lambda x.[x+1])(P_3) \\ P_3 = wp(\lambda x.[x\leq 100])(P_2) \vee wp(\lambda x.[x>100])(P_4) \\ P_4 = \phi \end{cases}$$

The merge over all paths and least fixpoint characterizations of the ascendants of the output states satisfying the exit specification $\phi=\lambda x.(x=101)$ are both equal to :

$$\left[\begin{array}{l} P_1 = \lambda x.(x\leq 101) \\ P_2 = \lambda x.(x\leq 100) \\ P_3 = \lambda x.(x\leq 101) \\ P_4 = \phi = \lambda x.(x=101) \end{array}\right.$$

End of Example.

In the following no distinction will be made between forward and backward program analyses because of the above mentioned symmetry.

4. APPROXIMATE ANALYSIS OF PROGRAMS

The semantic analysis of programs cannot be automatized since neither the merge over all paths nor the least fixpoint characterization of the invariant assertions to be generated leads to a computable function. Therefore optimizing compilers and program verification systems are only concerned with the discovery of approximate invariants assertions. Here an approximate invariant assertion Q will be one which is implied by the exact invariant assertion P defined by the deductive semantics.

DEFINITION 4.0.1

If $P,Q \epsilon A$ then "Q approximate P" iff $P \Rightarrow Q$.

This definition of "approximate" is the one which is useful in logical analysis of programs, data type determination and data flow analysis.(The dual one might be useful (e.g. for proving termination)).

The now classical lattice theoretic approach to approximate analysis of programs can be briefly sketched as follows : the representation of an approximate assertion is an element of a complete lattice $A(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$. The meaning of the elements of A is specified by a (too often implicit) order morphism γ mapping A to a subset of assertions $\overline{A}=\gamma(A)\subseteq A$. The intention is that A is an implementable image of those aspects $\gamma(A)$ of the program properties which are to be understood at each program point whereas the assertions belonging to $A-\gamma(A)$ are ignored (that is approximated from above in $\gamma(A)$). To each elementary command $S\epsilon L$ is associated an isotone map $t(S)$ from A to A. The intent is that $t(S)$ is an approximate predicate transformer such that $t(S)(i)$ represents the propagation of the information $i\epsilon A$ through the statement S.

The ideal merge over all paths program-wide analysis (Graham & Wegman[76], Kam & Ullman[77], Rosen[78], Tarjan[76]) is often approximated by a fixpoint solution (Cousot & Cousot[77a], Jones & Muchnick[76], Kaplan & Ullman[78], Kildall[73], Tenenbaum[74], Wegbreit[75]). A fixpoint system of isotone equations $X=F(X)$ where $F \epsilon (A^n \to A^n)$ is associated with the program graph. The approximate invariant assertions are generated by computing iteratively the least fixpoint of F starting from the infimum of A^n and using any chaotic or asynchronous iteration strategy (Cousot [77]) or the least fixpoint is approximated from above using an extrapolation technique in order to accelerate the convergence of the iterates whenever A does not satisfy the ascending chain condition (Cousot & Cousot[77a]).

The design of A, t, the implicit γ and the determination of the construction rules for F are often empirical. The correctness of the least fixpoint analysis is usually proved with respect to the approximate merge over all paths analysis, the correctness of which is taken for granted. As opposed to this empirical approach we now provide a formal approach to the systematic design of an approximate program analysis framework (A,t,γ) given $(V,\mathcal{A},\tau)$ where τ is *sp* for forward and *wp* for backward program analyses.

5. DESIGN OF A SPACE OF APPROXIMATE ASSERTIONS

5.1 A Very Reasonable Assumption

Assume that for a specific-purpose analysis of programs a subset $\overline{A}\subseteq\mathcal{A}$ of assertions has been found to provide meaningful information.

Since any invariant assertion $P\in\mathcal{A}$ for any program must have an upper approximation Q in $\overline{A}$, the set $\{Q\in\overline{A} : P\Rightarrow Q\}$ must be non empty.

Let $P\in\mathcal{A}$ be an assertion and assume that we want to analyze a program π using the merge over all paths semantic analysis and an entry condition Q which is an upper approximation of P in $\overline{A}$. What is the best choice for Q ? It is clear that if $P\Rightarrow Q'\Rightarrow Q$ then $F_\pi(\tau,Q')\Rightarrow F_\pi(\tau,Q)$ and by isotony the analysis $lfp(F_\pi(\tau,Q'))$ is more precise than $lfp(F_\pi(\tau,Q))$. Hence Q must be a minimal upper approximation of P in $\overline{A}$ (that is such that $\{P\Rightarrow Q \wedge \neg(\exists Q'\in\overline{A}-\{Q\} : P\Rightarrow Q' \wedge Q'\Rightarrow Q)\}$). Assume that the set U of minimal upper approximations of P in $\overline{A}$ has a cardinality greater than 1. What is the "best" possible choice for Q in U ? If $Q_1,Q_2\in U$ and $Q_1\neq Q_2$ then Q_1 and Q_2 are not necessarily comparable so that $lfp(F_\pi(\tau,Q_1))$ and $lfp(F_\pi(\tau,Q_2))$ may be not comparable. Hence "best" cannot be defined using the preciseness criterion provided by the ordering $\Rightarrow$. The only way to determine which of the two alternatives will be the most useful in order to answer a given set of application dependent questions about the program is to try both of them. Also the best choice may vary from one program to another. This try and see choice method leads to a non-determinist analysis method which is unacceptable because of obvious efficiency considerations. Therefore it is reasonable to choose $\overline{A}$ such that $Card(U)=1$.

Example 5.1.0.1

Assume that $\mathcal{A}=(Z\times Z)\rightarrow\mathcal{B}$ where Z is the set of integers and $\overline{A}=\{\lambda(x,y).[P_x(x)\wedge P_y(y)] : P_x,P_y \in \{\lambda u.\mathit{false},\lambda u.u\geq 0,\lambda u.u\leq 0,\lambda u.\mathit{true}\}\}$. The assertion $P=\lambda(x,y).(x\geq 0 \wedge y=0)$ has two distinct minimal upper approximations in $\overline{A}$ namely $Q_1=\lambda(x,y).(x\geq 0 \wedge y\geq 0)$ and $Q_2=\lambda(x,y).(x\geq 0 \wedge y\leq 0)$. Now the choice of the most useful upper approximation of the entry assertion P is program-dependent. For example the best choice is Q_1 for the program x:=x+y. This positive declaration can only be justified by performing the two semantic analyses (i.e. $sp(x{:=}x{+}y)(Q_1)=\lambda(x,y).(x\geq y \wedge y\geq 0)$ and $sp(x{:=}x{+}y)(Q_2)=\lambda(x,y).(x\geq y \wedge y\leq 0)$) and next comparing them. Since these analyses are not related by the ordering $\Rightarrow$, the comparison criterion must be application dependent. For example using Q_1 we can prove that $sp(\lambda x.x{+}y)(Q)\Rightarrow\lambda(x,y).(x\geq 0)$ whereas this is impossible with Q_2. On the contrary the best choice is Q_2 for the program x:=-x; x:=x+y since $sp(x{:=}{-}x;\ x{:=}x{+}y)(Q_2)=\lambda(x,y).(x\leq y \wedge y\leq 0)$ which implies $\lambda(x,y).(x\leq 0)$ whereas $sp(x{:=}{-}x;\ x{:=}x{+}y)(Q_1)=\lambda(x,y).(x\leq y \wedge y\geq 0)$ does not imply $\lambda(x,y).(x\leq 0)$.

End of Example.

If any program must have an analysis which can be approximated from above using $\overline{A}$, and the process for deriving the most useful approximate analysis of any program is required to be determinist then it is reasonable to make the following :

ASSUMPTION 5.1.0.2

The set $\overline{A}\subseteq\mathcal{A}$ of approximate assertions must be chosen such that for all $P\in\mathcal{A}$ the set $\{Q\in\overline{A} : P\Rightarrow Q\}$ of upper approximations of P in $\overline{A}$ has a least element.

THEOREM 5.1.0.3

For all $P\in\mathcal{A}$ the set $\{Q\in\overline{A} : P\Rightarrow Q\}$ has a least element if and only if $\overline{A}$ is a *Moore family* (i.e. $\overline{A}$ contains the supremum of $\mathcal{A}$ and is closed under conjunction).

5.2 The Approximation Operator

DEFINITION 5.2.0.1 *Approximation Operator*

$\rho \in \mathcal{A}\rightarrow\overline{A}$

$\rho = \lambda P.\wedge\{Q\in\overline{A} : P\Rightarrow Q\}$

$\rho(P)$ is the least upper approximation of P in $\overline{A}$. Since $\overline{A}$ is a Moore family it follows from Monteiro & Ribeiro[42,Th.5.3 and 5.1] that :

THEOREM 5.2.0.2

(1) - ρ is an upper closure operator (that is ρ is isotone (if $P,Q\in\mathcal{A}$ and $P\Rightarrow Q$ then $\rho(P)\Rightarrow\rho(Q)$), extensive (for all $P\in\mathcal{A}$, $P\Rightarrow\rho(P)$) and idempotent $(\rho=\rho\circ\rho)$

(2) - $\rho(\mathcal{A}) = \overline{A}$

(3) - ρ is the unique upper closure operator on $\mathcal{A}$ such that $\rho(\mathcal{A})=\overline{A}$

Since $\overline{A}$ is equal to the image of the complete lattice $\mathcal{A}(\Rightarrow,\lambda X.\mathit{false},\lambda X.\mathit{true},\vee,\wedge)$ by the upper closure operator ρ we derive from Ward[42,Th.4.1] the following :

THEOREM 5.2.0.3

(1) - $\overline{A}$ is a complete lattice $\rho(\mathcal{A})(\Rightarrow,\rho(\lambda X.\mathit{false}),\lambda X.\mathit{true},\lambda S.\rho(\vee S),\wedge)$

(2) - ρ is a quasi-complete join-morphism (i.e. $\forall S\subseteq\mathcal{A},\ \rho(\vee S) = \rho(\vee\rho(S))$)

(3) - $\overline{A}$ is a complete sub-lattice of $\mathcal{A}$ iff ρ is a complete join-morphism (i.e. $\forall S\subseteq\mathcal{A},\ \rho(\vee S)=\vee\rho(S)$)

If the initial choice of $\overline{A}$ does not satisfy assumption 5.1.0.2 we can use the following :

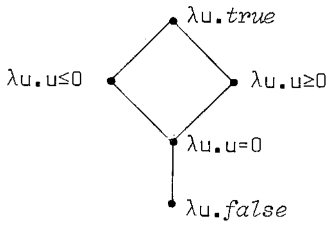
THEOREM 5.2.0.4

If $\overline{A} \subseteq A$, the upper closure operator ρ on A such that $\rho(A)$ is the least Moore family containing $\overline{A}$ is :

$\rho = \lambda P.\wedge\{Q \in \{\overline{A} \cup \{\lambda X.true\}\} : P \Rightarrow Q\}$
$\rho(A) = \wedge\{S : S \subseteq \{\overline{A} \cup \{\lambda X.true\}\} \wedge S \neq \emptyset\}$

Example 5.2.0.5

Returning to example 5.1.0.1 where $A = (Z \rightarrow B)$ and $\overline{A} = \{\lambda u.false, \lambda u.u \geq 0, \lambda u.u \leq 0, \lambda u.true\}$ the least Moore family containing $\overline{A}$ is the one containing $\lambda u.true$, $\overline{A}$ and the meets of the non-empty subsets of $\overline{A}$ that is the complete lattice :



The corresponding approximation operator is :

$\rho = \lambda P.$*if* $P = \lambda u.false$ *then* $\lambda u.false$
elsif $P \Rightarrow \lambda u.u=0$ *then* $\lambda u.u=0$
elsif $P \Rightarrow \lambda u.u \geq 0$ *then* $\lambda u.u \geq 0$
elsif $P \Rightarrow \lambda u.u \leq 0$ *then* $\lambda u.u \leq 0$
else $\lambda u.true$ *fi*

End of Example.

5.3 Representation of the Lattice of Approximate Assertions

In order to represent the approximate assertions in a computer memory we must use a complete lattice $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ such that the similar algebras $\overline{A} = \rho(A)(\Rightarrow, \lambda X.false, \lambda X.true, \lambda S.\rho(\vee S), \wedge)$ and $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be isomorphic. Let $\gamma \in (A \rightarrow \overline{A})$ be the corresponding lattice isomorphism. Let $\alpha \in (A \rightarrow A)$ be $\gamma^{-1} \circ \rho$. $\alpha(P)$ is the representation of the least upper approximation of the assertion $P \in A$ whereas $\gamma(Q)$ provides the meaning of $Q \in A$. The connection $\langle\alpha, \gamma\rangle$ between A and A has the following property :

DEFINITION 5.3.0.1

Let $L_1(\sqsubseteq_1)$ and $L_2(\sqsubseteq_2)$ be posets. $\langle\alpha, \gamma\rangle$ is a *pair of adjoined functions* if and only if :

- $\alpha \in (L_1 \rightarrow L_2)$ is isotone
- $\gamma \in (L_2 \rightarrow L_1)$ is isotone
- $\forall x \in L_1, \forall y \in L_2, \{x \sqsubseteq_1 \gamma(y)\} \Longleftrightarrow \{\alpha(x) \sqsubseteq_2 y\}$

(Contrary to Scott[72]'s definition, L_1 and L_2 are not required to be continuous lattices and α, γ need not be continuous).

THEOREM 5.3.0.2

If ρ is an upper closure operator on A, the image $\gamma(A)$ of $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ through the lattice isomorphism γ is equal to $\rho(A)(\Rightarrow, \rho(\lambda X.false), \lambda X.true, \lambda S.\rho(\vee S), \wedge)$ and $\alpha = \gamma^{-1} \circ \rho$ then

- $\langle\alpha, \gamma\rangle$ is a pair of adjoined functions
- α is onto, γ is one-to-one

Reciprocally the approximation process can be defined by the lattice $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ and a pair of adjoined functions. Such a pair $\langle\alpha, \gamma\rangle$ defines a Galois connection between A and the dual of A :

DEFINITION 5.3.0.3

Let $L_1(\sqsubseteq_1)$ and $L_2(\sqsubseteq_2)$ be posets, $\alpha \in (L_1 \rightarrow L_2)$, $\gamma \in (L_2 \rightarrow L_1)$. The pair $\langle\alpha, \gamma\rangle$ defines a *Galois connection* between L_1 and L_2 if and only if :

(1) - α is antitone $(\forall x_1, x_2 \in L_1, \{x_1 \sqsubseteq_1 x_2\} \Rightarrow \{\alpha(x_1) \sqsupseteq_2 \alpha(x_2)\})$
(2) - γ is antitone $(\forall y_1, y_2 \in L_2, \{y_1 \sqsubseteq_2 y_2\} \Rightarrow \{\gamma(y_1) \sqsupseteq_1 \gamma(y_2)\})$
(3) - $\lambda x \in L_1.(x) \sqsubseteq_1 \gamma \circ \alpha$
(4) - $\lambda y \in L_2.(y) \sqsubseteq_2 \alpha \circ \gamma$

The above conditions (3) and (4) are equivalent to : $\forall x \in L_1, \forall y \in L_2, \{x \sqsubseteq_1 \gamma(y)\} \Longleftrightarrow \{\alpha(x) \sqsupseteq_2 y\}$ (Birkhoff[67]), hence we have :

THEOREM 5.3.0.4

Let $L_1(\sqsubseteq_1)$, $L_2(\sqsubseteq_2)$ be posets, $\alpha \in (L_1 \rightarrow L_2)$, $\gamma \in (L_2 \rightarrow L_1)$. $\langle\alpha, \gamma\rangle$ is a pair of adjoined functions if and only if $\langle\alpha, \gamma\rangle$ defines a Galois connection between $L_1(\sqsubseteq_1)$ and $L_2^{\partial}(\sqsubseteq_2^{\partial}) = L_2(\sqsupseteq_2)$, (i.e. iff α and γ are isotone, $\lambda x.x \sqsubseteq_1 \gamma \circ \alpha$, $\alpha \circ \gamma \sqsubseteq_2 \lambda y.y$)

Theorem 5.3.0.4, Ore[44,Th.2] and Pickert[52] imply :

COROLLARY 5.3.0.5

Let $L_1(\sqsubseteq_1)$ and $L_2(\sqsubseteq_2)$ be posets and $\alpha \in (L_1 \rightarrow L_2)$, $\gamma \in (L_2 \rightarrow L_1)$ be adjoined functions :

(1) - $\gamma \circ \alpha$ is an upper closure operator on L_1, $\alpha \circ \gamma$ is a lower closure operator on L_2 (i.e. isotone, reductive $(\alpha \circ \gamma \sqsubseteq \lambda x.x)$ and idempotent)

Moreover if $L_1(\sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\sqsubseteq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ are complete lattices then :

(2) - $\gamma \circ \alpha(L_1)$ and $\alpha \circ \gamma(L_2)$ are complete lattices. α is an isomorphism from $\gamma \circ \alpha(L_1)$ onto $\alpha \circ \gamma(L_2)$ and γ is an isomorphism from $\alpha \circ \gamma(L_2)$ onto $\gamma \circ \alpha(L_1)$
(3) - Each function in the pair $\langle\alpha, \gamma\rangle$ of adjoined functions uniquely determines the other, more precisely :
(3.1) - $\alpha = \lambda x \in L_1.\sqcap_2\{y \in L_2 : x \sqsubseteq_1 \gamma(y)\}$
(3.2) - $\gamma = \lambda y \in L_2.\sqcup_1\{x \in L_1 : \alpha(x) \sqsubseteq_2 y\}$
(4) - α is a complete join-morphism, $\alpha(\bot_1) = \bot_2$, γ is a complete meet-morphism, $\gamma(\top_2) = \top_1$

In complement we will need the following :

THEOREM 5.3.0.6

Let $L_1(\sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and $L_2(\sqsubseteq_2, \bot_2, \top_2, \sqcup_2, \sqcap_2)$ be complete lattices and $\alpha \in (L_1 \rightarrow L_2)$, $\gamma \in (L_2 \rightarrow L_1)$ be adjoined functions :

(1) - α is onto (surjective) if and only if γ is one-to-one (injective) and if and only if $\alpha \circ \gamma = \lambda y \in L_2.(y)$
(2) - if one of the above conditions holds then

- $\gamma = \lambda y \in L_2 . \sqcup_1 \{x \in L_1 : \alpha(x) = y\}$
- α is an isomorphism from the complete lattice $\gamma \circ \alpha(L_1)$ onto the complete lattice L_2 the inverse of which is γ.

(3) - α is one-to-one if and only if γ is onto and if and only if $\gamma \circ \alpha = \lambda x \in L_1 . (x)$

We use the notation $L_1 \rhd\!\!< \alpha, \gamma > L_2$ to state that L_1 and L_2 are connected by the pair $<\alpha,\gamma>$ of adjoined functions which are respectively surjective and injective. If α is a complete join-morphism from L_1 onto L_2 (respectively γ is a one-to-one complete meet-morphism from L_2 into L_1) we write $L_1 \rhd\!\!< \alpha, > L_2$ ($L_1 \rhd\!\!< , \gamma > L_2$) and assume that the adjoined $\gamma(\alpha)$ is determined by 5.3.0.5.(3.2) (5.3.0.5.(3.1)).

In the literature the most usual method for defining a program analysis framework is to specify the complete lattice $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ representing approximate assertions and to informally describe the meaning of its elements (e.g. constant propagation, Kildall[73], Kam & Ullman[77]). Hence the function $\gamma \in (A \to \mathcal{A})$ remains implicit.

It is often the case that A is only assumed to be a (complete) join-semi-lattice $A(\sqsubseteq, \top, \sqcup)$ (or dually meet-semi-lattice for some authors) but since an infimum is adjoined to A it is in fact a complete lattice (even when the meet-operation is not used or what is called meet is not $\sqcap$ (e.g. Wegbreit[75])).

When $\gamma \in (A \to \mathcal{A})$ is isotone but not a complete meet-morphism the set $\gamma(A)$ does not fulfill assumption 5.1.0.2 with the consequences examined at paragraph 5.1. The design of $\gamma(A)$ and A can be revised as stated by theorem 5.2.0.4.

When $\gamma \in (A \to \mathcal{A})$ is a complete meet-morphism but not one-to-one, several distinct elements of A have the same meaning. Since this is useless, the design of A and γ can be revised as follows :

THEOREM 5.3.0.7

Let $A(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice and $\gamma \in (A \to \mathcal{A})$ be a complete meet-morphism. Let $\sigma \in (A \to A)$ be $\lambda x . \sqcap \{y \in A : \gamma(x) = \gamma(y)\}$, $\tilde{A} = \sigma(A)$, $\tilde{\gamma} = (\gamma | \tilde{A})$:

- $\forall x \in A,\ \gamma(x) = \gamma(\sigma(x))$
- σ is a lower closure operator on A
- $\tilde{\gamma}$ is a one-to-one complete meet-morphism from the complete lattice $\tilde{A}(\sqsubseteq, \bot, \sigma(\top), \sqcup, \lambda S . \sigma(\sqcap S))$ into $\mathcal{A}$

Since $\gamma(A) = \tilde{\gamma}(\tilde{A})$, A and $\tilde{A}$ have the same expressive power. Among all subsets of A which have the expressive power of A, $\tilde{A}$ is one with minimal cardinality.

THEOREM 5.3.0.8

(1) - $\forall L \subseteq A,\ \{\gamma(L) = \gamma(A)\} \Rightarrow \{\sigma(L) = \tilde{A}\}$
(2) - $\forall L \subseteq A,\ \{\gamma(L) = \gamma(A)\} \Rightarrow \{\mathrm{Card}(\tilde{A}) \le \mathrm{Card}(L)\}$
(3) - $\forall x \in A,\ \forall y \in \tilde{A},\ \{\gamma(x) = \tilde{\gamma}(y)\} \Rightarrow \{y \sqsubseteq x\}$

6. EQUIVALENT METHODS FOR SPECIFYING A SPACE OF APPROXIMATE ASSERTIONS

A space of approximate assertions can be specified either by a Moore family or by an upper closure operator. Moore families can be characterized using definition 5.1.0.2 or theorems 5.1.0.3 and 5.2.0.4. In addition to theorems 5.2.0.2.(1) and 5.3.0.6 we now study and examplify various equivalent methods which can be used to define an upper closure operator.

6.1 Least Closure Operator Greater than or Equal to an Arbitrary Function

THEOREM 6.1.0.1

Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice and $f \in (L \to L)$.

- Let $iso \in ((L \to L) \to (L \to L))$ be $\lambda f . [\lambda x . \sqcup \{f(y) : y \sqsubseteq x\}]$. $iso(f)$ is the least isotone operator on L greater than or equal to f
- Let $ext \in ((L \to L) \to (L \to L))$ be $\lambda f . [\lambda x . [x \sqcup f(x)]]$. $ext(f)$ is the least extensive operator on L greater than or equal to f
- Let $ide \in ((L \to L) \to (L \to L))$ be $\lambda x . [luis(f)(x)]$ where $luis(f)(x)$ is the limit of the increasing and ultimately constant sequence $\{X^\delta\}$ such that $X^0 = x$, for every ordinal δ, $X^{\delta+1} = f(X^\delta)$ and for every limit ordinal δ, $X^\delta = \bigsqcup_{\alpha<\delta} X^\alpha$
- $clo(f) = ide(ext(iso(f)))$ is the least closure operator greater than or equal to f and $clo(f)(L)$ is the greatest Moore family contained in $f(L)$

6.2. Definition of a Space of Approximate Assertions by Composition of Upper Closure Operators

The composition of two upper closure operators on $\mathcal{A}$ is usually not a closure operator (Ore[43]). However the space of approximate assertions can be designed by successive approximations using the following composition of upper closure operators :

THEOREM 6.2.0.1

Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice, ρ an upper closure operator on L and η be an upper closure operator on $\rho(L)$. Then $\eta \circ \rho$ is an upper closure operator on L and $\rho \sqsubseteq \eta \circ \rho$.

Example 6.2.0.2

Many program analysis frameworks are designed in order to describe some properties of each program variable but so that the relationships among the values of these variables are ignored. An example is Jones & Muchnick[76]'s type determination scheme, a counter-example is the determination of linear relationships among numerical variables, Cousot & Halbwachs [78]. The corresponding approximation can be characterized as follows :

Assume that $\mathcal{V} = \mathcal{D}^m$, let A_m be $(\mathcal{V} \to \mathcal{B})$ and A_1 be $(\mathcal{D} \to \mathcal{B})$. Let us define :

$$\forall j \in [1,m],\ \sigma_j \in (A_m \to A_1)$$
$$\sigma_j = \lambda P \in A_m . [\lambda x \in \mathcal{D} . [\exists <v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_m> \in \mathcal{D}^{m-1} : P(v_1, \ldots, v_{j-1}, x, v_{j+1}, \ldots, v_m)]]$$
$$\sigma \in (A_m \to A_m)$$
$$\sigma = \lambda P \in A_m . [\lambda (x_1, \ldots, x_m) \in \mathcal{D}^m . [\bigwedge_{j=1}^{m} \sigma_j(P)(x_j)]]$$

σ is an upper closure operator on A_m and an assertion $P \in A_m$ does not state relationships among the program variables if and only if $\sigma(P)=P$. The approximate assertions on each individual program variable x_j are next defined using an upper closure operator ρ_j on A_1. The induced closure operation ρ on $\sigma(A_m)$ is defined by $\rho(P) = \lambda(x_1,\ldots,x_m).\bigwedge_{j=1}^{m}\rho_j(P_j)(x_j)$ where $P \in \sigma(A_m)$ is (necessarily) of the form $P = \lambda(x_1,\ldots,x_m).\bigwedge_{j=1}^{m} P_j(x_j)$. It follows from theorem 6.2.0.1 that the composition :

$$\rho\circ\sigma = \lambda P \in A_m.[\lambda(x_1,\ldots,x_m)\in \mathcal{D}^m.[\bigwedge_{j=1}^{m}\rho_j(\sigma_j(P))(x_j)]]$$

is an upper closure operator on A_m.

End of Example.

6.3 Definition of a Space of Approximate Assertions by Means of a Complete Join Congruence Relation

Considering the equivalence relation (ρ) induced by an upper closure operator ρ on A and defined as $P \equiv Q(\rho)$ if and only if $\rho(P)=\rho(Q)$, the approximation process can be understood as essentially consisting in partitionning the space of assertions so that no distinction is made between equivalent assertions which are all approximated by a representant of their equivalence class. Since the approximation is from above and a least one must exist (assumption 5.1.0.2) not all equivalence relations are acceptable :

DEFINITION 6.3.0.1

Let $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ be a complete lattice. A binary relation θ on L is a *complete join-congruence relation* if and only if :

(1) - θ is an equivalence relation

(2) - θ satisfies the *join-substitution property* : $\forall x,y,u \in L,\ x \equiv y(\theta) \Rightarrow x \sqcup u \equiv y \sqcup u(\theta)$

(3) - θ satisfies the *join-completeness property* : $\forall x \in L,\ x \equiv \sqcup[x]\theta\ (\theta)$ where $[x]\theta = \{y \in L : x \equiv y(\theta)\}$ is the congruence class containing x.

THEOREM 6.3.0.2

If ρ is an upper closure operator on $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ and $\forall x,y \in L,\ x \equiv y(\rho)$ if and only if $\rho(x)=\rho(y)$ then :

(1) - (ρ) is a complete join-congruence relation on L

(2) - $\rho = \lambda x.\sqcup([x](\rho))$

Reciprocally a complete join-congruence relation on A defines an upper closure operator on A whence a space of approximate assertions :

THEOREM 6.3.0.3

Let θ be a complete join-congruence relation on the complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$. $\lambda x.\sqcup([x]\theta)$ is an upper closure operator on L.

(Similar results were already proved in Cousot & Cousot[77b] except that the above definition of complete join-congruence relations has been substantially simplified).

The following result can sometimes facilitate the proof that a given relation is a join-congruence relation (satisfying 6.3.0.1.(1) and 6.3.0.1.(2)). It can be compared with Grätzer & Schmidt[58]'s theorem which is relative to congruence relations.

THEOREM 6.3.0.4

A reflexive and symmetric binary relation θ on a complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is a join-congruence relation iff the following three properties are satisfied for $x,y,z,t \in L$:

(1) - $\{x \equiv y(\theta)\} \Leftrightarrow \{\exists u \in L : (x \sqcup y) \sqsubseteq u \wedge u \equiv x(\theta) \wedge u \equiv y(\theta)\}$

(2) - $\{x \sqsubseteq y \sqsubseteq z \wedge x \equiv y(\theta) \wedge y \equiv z(\theta)\} \Rightarrow \{x \equiv z(\theta)\}$

(3) - $\{x \sqsubseteq y \wedge x \equiv y(\theta)\} \Rightarrow \{(x \sqcup t) \equiv (y \sqcup t)(\theta)\}$

Example 6.3.0.5

Let V be a non-empty set of integers included between two bounds $-\infty$ and $+\infty$ (either $V = Z \cup \{-\infty,+\infty\}$ and $\forall i \in Z,\ -\infty \le -\infty < i < +\infty \le +\infty$ or $\{-\infty,+\infty\} \subseteq Z$ and $V = \{i \in Z : -\infty \le i \le +\infty\}$). The binary relation θ defined on $A = (V \to B)$ by :

$$\{P \equiv Q(\theta)\} \Leftrightarrow \{min\{x \in V : P(x)\} = min\{x \in V : Q(x)\} \wedge max\{x \in V : P(x)\} = max\{x \in V : Q(x)\}\}$$

(where $min(\emptyset) = +\infty$ and $max(\emptyset) = -\infty$) is a complete join congruence relation. The quotient lattice L/θ is isomorphic to $\rho(A)$ where ρ is the upper closure operator induced by θ :

$$\rho = \lambda P.\vee([P]\theta) = \lambda P \in A.[\lambda x \in V.[min\{y : P(y)\} \le x \le max\{y : P(y)\}]]$$

In conjunction with 6.2.0.2, ρ can be used for static analysis of the ranges of values of numerical variables (Cousot & Cousot[77a]).

End of Example.

6.4. Definition of a Space of Approximate Assertions by Means of a Family of Principal Ideals

The equivalence classes of the complete join-congruence relation (ρ) induced by a closure operator ρ have the following property :

THEOREM 6.4.0.1

Let θ be a complete join-congruence relation on the complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$, then $\forall x \in L$, $[x]\theta$ is a complete and convex sub-join-semilattice of L. (Let us recall that $S \subseteq L$ is *convex* iff $a,b \in S$, $c \in L$ and $a \sqsubseteq c \sqsubseteq b$ imply that $c \in S$)

Here is another representation of convex sub-join-semilattices of L (which can be compared with Grätzer[71]'s representation of convex sublattices) :

An *ideal* is a nonvoid subset J of a lattice $L(\sqsubseteq,\bot,\sqcup)$ with the properties (a) $\{\{a \in J, x \in L, x \sqsubseteq a\} \Rightarrow \{x \in J\}\}$ and (b) $\{\{a \in J, b \in J\} \Rightarrow \{a \sqcup b \in J\}\}$. It is easy to show that J is an ideal when $(a \sqcup b) \in J$ holds if and only if $a \in J$ and $b \in J$ (Caste property). Since L has an infimum $\bot$, the intersection of an infinite family of ideals in a lattice L is an ideal of L.

Given an element a in a lattice L, the set $\{x \in L : x \sqsubseteq a\}$ is evidently an ideal; it is called a *principal ideal* of L. If every ascending chain in L is finite, every ideal is principal.

A *semi-ideal* is a nonvoid subset I of L with the property $\{\{a \in I, x \in L, x \sqsubseteq a\} \Rightarrow \{x \in I\}\}$. The dual notion is the one of *dual semi-ideal*.

THEOREM 6.4.0.2

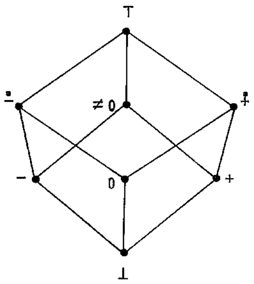
(1) - Let I be a principal ideal and J be a dual semi-ideal of a complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$. If $I \cap J$ is nonvoid then $I \cap J$ is a complete and convex sub-join-semilattice of L.
(2) - Every complete and convex sub-join-semilattice C of L can be expressed in this form with $I = \{x \in L : x \sqsubseteq (\sqcup C)\}$ and $\{x \in L : \{\exists y \in C : y \sqsubseteq x\}\} \subseteq J$.

THEOREM 6.4.0.3

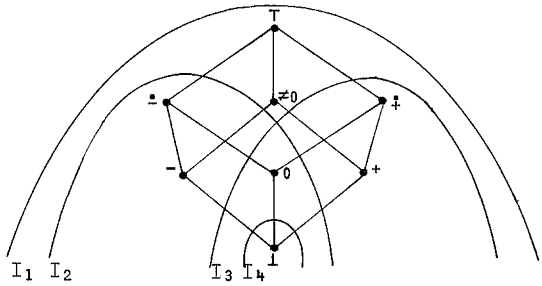
Let $\{I_i \in \Delta\}$ be a family of principal ideals of the complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ containing L. Then $\lambda x.\sqcup\{\cap I_i : i \in \Delta \wedge x \in I_i\}$ is an upper closure operator on L.

Example 6.4.0.4

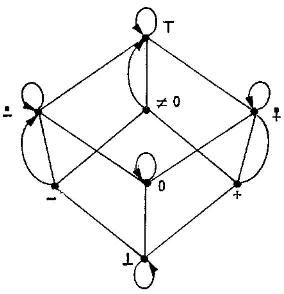
The following lattice can be used for static analysis of the signs of values of numerical variables :



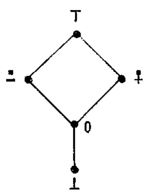
(where $\bot$, $-$, $+$, $\dot{-}$, $\neq 0$, $\dot{+}$, $\top$ respectively stand for $\lambda x.$*false*, $\lambda x.x<0$, $\lambda x.x>0$, $\lambda x.x\leq 0$, $\lambda x.x\neq 0$, $\lambda x.x\geq 0$, $\lambda x.$*true*). A further approximation can be defined by the following family of principal ideals :



which induces an upper closure operator ρ :



and the space of approximate assertions (used in example 5.2.0.5)



End of Example.

7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to A and γ the specification of a program analysis framework also includes the choice of an approximate predicate transformer $t \in (L \to (A \to A))$ (or a monoid of maps on A plus a rule for associating maps to program statements (e.g. Rosen[78])). We now show that in fact this is not indispensable since there exists a best correct choice of τ which is induced by $\bar{A}$ and the formal semantics of the considered programming language.

7.1 A Reasonable Definition of Correct Approximate Predicate Transformers

At paragraph 3, given (V,A,τ) the minimal assertion which is invariant at point i of a program π with entry specification $\phi \in A$ was defined as :

$$P_i = \bigvee_{p \in path(i)} \tilde{\tau}(p)(\phi)$$

Therefore the minimal approximate invariant assertion is the least upper approximation of P_i in $\bar{A}$ that is :

$$\rho(P_i) = \rho(\bigvee_{p \in path(i)} \tilde{\tau}(p)(\phi))$$

Even when path(i) is a finite set of finite paths the evaluation of $\tilde{\tau}(p)(\phi)$ is hardly machine-implementable since for each path $p = a_1,\ldots,a_m$ the computation sequence $X_0 = \phi$, $X_1 = \tau(C(a_1))(X_0)$, ..., $X_m = \tau(C(a_m))(X_{m-1})$ does not necessarily only involve elements of $\bar{A}$ and $(\bar{A}\to\bar{A})$. Therefore using $\bar{\phi}\in\bar{A}$ and $t \in (L \to (\bar{A}\to\bar{A}))$ a machine representable sequence $\bar{X}_0=\bar{\phi}$, $\bar{X}_1=\bar{t}(C(a_1))(\bar{X}_0)$, ..., $\bar{X}_m=\bar{t}(C(a_m))(\bar{X}_{m-1})$ is used instead of $X_0,\ldots,X_m$ which leads to the expression :

$$Q_i = \rho(\bigvee_{p \in path(i)} \tilde{\bar{t}}(p)(\bar{\phi}))$$

The choice of $\bar{t}$ and $\bar{\phi}$ is correct if and only if Q_i is an upper approximation of P_i in $\bar{A}$ that is if and only if :

$$(\bigvee_{p \in path(i)} \tilde{\tau}(p)(\phi)) \Rightarrow \rho(\bigvee_{p \in path(i)} \tilde{\bar{t}}(p)(\bar{\phi}))$$

In particular for the entry point we must have $\phi \Rightarrow \rho(\bar{\phi})=\bar{\phi}$ so that we can state the following :

DEFINITION 7.1.0.1

(1) - An approximate predicate transformer $\bar{t}\in(L\to(\bar{A}\to\bar{A}))$ is said to be a *correct upper approximation* of $\tau\in(L\to(A\to A))$ in $\bar{A}=\rho(A)$ if and only if for all $\phi\in A$, $\bar{\phi}\in\bar{A}$ such that $\phi\Rightarrow\bar{\phi}$ and program π we have : $MOP_\pi(\tau,\phi)\Rightarrow MOP_\pi(\bar{t},\bar{\phi})$
(2) - Similarly if $A \triangleright\!\!<\alpha,\gamma> A$, $t\in(L\to(A\to A))$ is said to be a *correct upper approximation* of $\tau\in(L\to(A\to A))$ in $A=\alpha(A)$ if and only if $\forall\phi$, $\forall\bar{\phi}$: $\phi\Rightarrow\gamma(\bar{\phi})$, $\forall\pi$, $\alpha(MOP_\pi(\tau,\phi)) \sqsubseteq MOP_\pi(\bar{t},\bar{\phi}))$, (i.e. $MOP_\pi(\tau,\phi) \Rightarrow \gamma(MOP_\pi(\bar{t},\bar{\phi}))$)

This global correctness condition for $\bar{t}$ is very difficult to check since for any program π and any program point i all paths $p\in path(i)$ must be considered. However it is possible to use instead the following equivalent local condition which can be checked for every type of statements :

THEOREM 7.1.0.2

(1) - $\overline{t} \in (L \to (\overline{A} \to \overline{A}))$ is a correct upper approximation of $\tau \in (L \to (A \to A))$ in $\overline{A}=\rho(A)$ iff $\{\forall S \in L,\ \forall P \in \overline{A},\ \tau(S)(P) \Rightarrow \overline{t}(S)(P)\}$

(2) - $t \in (L \to (A \to A))$ is a correct upper approximation of $\tau \in (L \to (A \to A))$ in $A=\alpha(A)$ (where $A \triangleright\!\!<\alpha,\gamma> A$) iff $\{\forall S \in L,\ \forall P \in A,\ \alpha(\tau(S)(\gamma(P))) \sqsubseteq t(S)(P)\}$.

If $\overline{t} \in (L \to (\overline{A} \to \overline{A}))$ is a correct upper approximation of $\tau \in (L \to (A \to A))$ in $\overline{A}=\rho(A)$ we have $MOP_\pi(\tau,\phi) \Rightarrow MOP_\pi(\overline{t},\rho(\phi))$ whence $\rho(MOP_\pi(t,\phi)) \Rightarrow MOP_\pi(\overline{t},\rho(\phi))$. The cases when equality holds are not easy to distinguish. Yet the following sufficient condition turns out to be useful afterwards :

THEOREM 7.1.0.3

(1) - If $\overline{t}$ is a correct upper approximation of τ in $\rho(A)$ and $\{\forall S \in L,\ \forall P \in A,\ \rho(\tau(S)(P)) = \overline{t}(S)(\rho(P))\}$ then $\forall \pi,\ \forall \phi,\ \rho(MOP_\pi(\tau,\phi)) = MOP_\pi(\overline{t},\rho(\phi))$

(2) - If t is a correct upper approximation of τ in A where $A \triangleright\!\!<\alpha,\gamma> A$ and $\{\forall S \in L,\ \forall P \in A,\ \alpha(\tau(S)(P)) = t(S)(\alpha(P))\}$ then $\forall \pi,\ \forall \phi,\ \alpha(MOP_\pi(\tau,\phi)) = MOP_\pi(t,\alpha(\phi))$

Similar results hold for fixpoint analysis of programs :

THEOREM 7.1.0.4

Let $t \in (L \to (A \to A))$ be an isotone correct upper approximation of $\tau \in (L \to (A \to A))$ in $A=\alpha(A)$ where $A \triangleright\!\!<\alpha,\gamma> A$ then $\forall \phi \in A,\ \forall \overline{\phi} \in A : \phi \Rightarrow \gamma(\overline{\phi}),\ \forall \pi,$

(1) - $F_\pi(t,\overline{\phi})$ is isotone and $\alpha \circ F_\pi(\tau,\phi) \circ \gamma \sqsubseteq F_\pi(t,\overline{\phi})$

(2) - $\alpha(\mathit{lfp}(F_\pi(\tau,\phi))) \sqsubseteq \mathit{lfp}(F_\pi(t,\overline{\phi}))$

(3) - If $\phi = \gamma(\overline{\phi})$ and $\{\forall S \in L,\ \forall P \in A,\ \alpha(\tau(S)(P)) = t(S)(\alpha(P))\}$ then equality holds in (2)

Notice that in theorem 7.1.0.2 the maps $\{\overline{t}(S) : S \in L\}$ are not assumed to be isotone. Yet isotony is assumed in theorem 7.1.0.4 and is a customary hypothesis in the literature. An apparent justification of this additional requirement is to ensure that the system of equations $X=F_\pi(t,\overline{\phi})(X)$ associated with a program π has fixpoints which can be obtained as limits of iteration sequences. But this could also be achieved without isotony hypothesis taking $\lambda X.X \sqcup F_\pi(t,\overline{\phi})(X)$ instead of $F_\pi(t,\overline{\phi})$:

THEOREM 7.1.0.5

Let $t \in (L \to (A \to A))$ be a correct upper approximation of $\tau \in (L \to (A \to A))$ in $A=\alpha(A)$ where $A \triangleright\!\!<\alpha,\gamma> A$ then $\forall \phi \in A,\ \forall \overline{\phi} \in A : \phi \Rightarrow \gamma(\overline{\phi}),\ \forall \pi,$

(1) - $\alpha \circ F_\pi(\tau,\phi) \circ \gamma \sqsubseteq F_\pi(t,\overline{\phi})$

(2) - $\alpha(\mathit{lfp}(F_\pi(\tau,\phi))) \sqsubseteq \mathit{luis}[\lambda X.X \sqcup F_\pi(t,\overline{\phi})(X)][\bot]$ (where $\mathit{luis}(F)(\bot)$ is the limit of the stationary iteration sequence $X^0=\bot$, $X^\delta=F(X^{\delta-1})$ for successor ordinals, $X^\delta = \bigsqcup_{\alpha<\delta} X^\alpha$ for limit ordinals)

Hence the isotony hypothesis is even not necessary for technical purposes. However the profound justification of this hypothesis can be found in the fact that among all possible approximate predicate transformers which can be used with a given set A of approximate assertions the designer of a program analysis framework intuitively thinks to the best approximate predicate transformer which happens to be isotone. This property also explains the fact that no significant counter-examples to the isotony hypothesis have ever been found.

7.2 The Best Approximate Predicate Transformer Induced by a Space of Approximate Assertions

DEFINITION 7.2.0.1

If $\overline{t}_1$, $\overline{t}_2$ are correct upper approximations of $\tau \in (L \to (A \to A))$ in $\overline{A}=\rho(A)$ then we say that $\overline{t}_1$ *is better than* $\overline{t}_2$ iff for all $\overline{\phi} \in \overline{A}$ and all programs π, $MOP_\pi(\overline{t}_1,\overline{\phi}) \Rightarrow MOP_\pi(\overline{t}_2,\overline{\phi})$

LEMMA 7.2.0.2

Let $\overline{t}_1$, $\overline{t}_2$ be correct upper approximations of $\tau \in (L \to (A \to A))$ in $\overline{A}=\rho(A)$. If $(\forall S \in L,\ \overline{t}_1(S) \Rightarrow \overline{t}_2(S))$ and ($\overline{t}_1$ or $\overline{t}_2$ is isotone) then $\overline{t}_1$ is better than $\overline{t}_2$. (Notice that the above isotony condition is sufficient but not necessary).

THEOREM 7.2.0.3

Let $\overline{\tau}$ be $\lambda S \in L.[\lambda P \in \overline{A}.[\rho(\tau(S)(P))]]$

(1) - $\forall S \in L,\ \overline{\tau}(S) \in (\overline{A} \to \overline{A})$ is isotone

(2) - $\overline{t} \in (L \to (\overline{A} \to \overline{A}))$ is a correct upper approximation of $\tau \in (L \to (A \to A))$ in $\overline{A}=\rho(A)$ iff $\{\forall S \in L,\ \overline{\tau}(S) \Rightarrow \overline{t}(S)\}$

(3) - $\overline{\tau}$ is the best correct upper approximation of τ in $\overline{A}$

COROLLARY 7.2.0.4

If $A \triangleright\!\!<\alpha,\gamma> A(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ then $\lambda S \in L.[\lambda P \in A.[\alpha(\tau(S)(\gamma(P)))]]$ is isotone, it is the best correct upper approximation of τ in A.

The most interesting consequence is that we have in hand a formal specification of the programs which have to be written in order to implement any specific program analysis framework once A and γ have been chosen. As a challenge to automatic program synthesizers let us consider a simple

Example 7.2.0.5

Coming back to examples 6.2.0.2 and 6.3.0.5 assume that $\mathcal{D}$ is the set of integers included between two bounds $-\infty$ and $+\infty$ and $V=\mathcal{D}^m$. For simplicity we shall assume that m=2. Let L be the complete lattice $\{\bot\} \cup \{[a,b] : a,b \in \mathcal{D} \wedge a \leq b\}$ with ordering $[a,b] \sqsubseteq [c,d]$ iff $a \leq b \leq c \leq d$ and $\bot$ is the infimum. Let $\gamma' \in (L \to (\mathcal{D} \to \mathcal{B}))$ be such that $\gamma'(\bot)=\lambda x.\mathit{false}$, $\gamma'([a,b])=\lambda x.(a \leq x \leq b)$. Let A be $L \Box L$ where $L \Box M = \{x \Box y : x \in L \wedge y \in M\}$ and $x \Box y$ = *if* $x=\bot$ or $y=\bot$ *then* $<\bot,\bot>$ *else* $<x,y>$ *fi*. Let $\gamma \in (A \to A)$. where $A=(\mathcal{D}^2 \to \mathcal{B})$ be $\lambda<x,y>.(\gamma'(x) \wedge \gamma'(y))$.

Given A and γ let us determine the best correct upper approximation of *sp* in A. Again for lack of space we just study the case of $sp(\lambda<x,y>.(x \leq y))$.

Since γ' is an injective complete meet-morphism the adjoined function $\alpha' \in ((\mathcal{D} \to \mathcal{B}) \to L)$ is determined by 5.3.0.5.(3.1) : $\alpha' = \lambda P.$ *if* $P=\lambda x.\mathit{false}$ *then* $\bot$ *else* $[\mathit{min}\{x : P(x)\}, \mathit{max}\{x : P(x)\}]$ *fi*. The same way, $\alpha \in (A \to L \Box L)$ is $\lambda P.(\alpha'(\sigma_x(P)) \Box \alpha'(\sigma_y(P)))$ where $\sigma_x = \lambda P.[\lambda x.[\exists y \in \mathcal{D} : P(x,y)]]$ and $\sigma_y = \lambda P.[\lambda y.[\exists x \in \mathcal{D} : P(x,y)]]$

According to theorem 7.2.0.4 the best upper approximation of $sp(\lambda\langle x,y\rangle.(x\leq y))$ in A is $t = \alpha\circ sp(\lambda\langle x,y\rangle.(x\leq y))\circ\gamma$. If $P\in A$ equals $\langle\bot,\bot\rangle$ then $t(P)=\langle\bot,\bot\rangle$ else $P=\langle[a,b],[c,d]\rangle$ where $a\leq b$ and $c\leq d$ in which case $t(P)=\alpha(Q)$ where $Q=\lambda\langle x,y\rangle.(a\leq x\leq b \wedge c\leq y\leq d \wedge x\leq y)$. $\sigma_x(Q)=\lambda x.(\exists y : a\leq x\leq b \wedge c\leq y\leq d \wedge x\leq y)=\lambda x.(a\leq x\leq b \wedge \mathit{max}(c,x)\leq d)=\lambda x.(a\leq x\leq b \wedge x\leq d)$ since $c\leq d$. The same way $\sigma_y(Q)=\lambda y.(\mathit{max}(c,a)\leq y\leq d)$. Therefore $t(P) = \mathit{if}\ a>d\ \mathit{then}\ \langle\bot,\bot\rangle\ \mathit{else}\ \langle[a,\mathit{min}(b,d)],[\mathit{max}(c,a),d]\rangle\ \mathit{fi} = \langle[a,b]\sqcap[-\infty,d],[c,d]\sqcap[a,+\infty]\rangle$ proving that this choice in Cousot & Cousot[77a] was optimal.

End of Example.

Example 7.2.0.6

Some program analyses (such as "reaching definitions", "available expressions", "live variables", ... Aho & Ullman[77]) are "history sensitive" because the approximate assertions which are useful at each program point p characterize sets of sequences of states (or execution paths from the entry point to p) and not sets of states. In such a case Hoare[78]'s formal definition of languages using sets of sequential traces is more convenient that the deductive semantics of paragraph 3.

7.2.0.6.1. *Associating a Set of Traces with a Program*

Given a universe V of values, a set L_a of elementary assignments, a set L_t of elementary tests, the set of *sequential traces* is the free monoid $T(;,\langle\rangle)$ generated by $L = L_a \cup L_t$.

The concatenation operation ";" is extended to elements of the complete lattice $2^T(\subseteq,\emptyset,T,\cup,\cap)$ by $S;T = \{s;t : s\in S \wedge t\in T\}$.

Let us define a forward "set of traces transformer" $ft \in (L\to(2^T\to 2^T))$ as $\lambda S.[\lambda T.[T;\{S\}]]$. The set of traces associated with a program π and an entry specification $\phi\in A\subseteq L_t$ is $\mathrm{MOP}_\pi(ft,\{\phi\})$.

7.2.0.6.2. *Approximating a Set of Traces by an Assertion Characterizing the Descendants of the Entry States*

The connection with the deductive semantics of paragraph 3 is made using $\alpha\in(2^T\to A)$ such that for any set T of traces, $\alpha(T)$ characterizes the possible descendants of the entry states (belonging to V) when the traces $t\in T$ are executed. From an (obvious hence not given here) operational semantics of sequential traces we derive that $\alpha=\lambda T.[\vee\{\alpha'(t) : t\in T\}]$ where $\alpha'\in(T\to A)$ is such that $\alpha'(\langle\rangle)=\lambda X\in V.\mathit{true}$ and $\forall S\in L$, $\forall t\in T$, $\alpha'(t;S)=sp(S)(\alpha'(t))$.

Since α is a complete join-morphism from 2^T onto A, theorem 5.3.0.5.(3.2) defines an adjoined function $\gamma\in(A\to 2^T)$.

According to theorem 7.2.0.4 the best correct upper approximation of ft in A is $\overline{ft} = \lambda S\in L.[\lambda P\in A.[\alpha(ft(S)(\gamma(P)))]]$. $\forall S\in L$, $\forall P\in A$, we have $\overline{ft}(S)(P)=\alpha(ft(S)(\gamma(P)))=\alpha(\gamma(P);\{S\})=\alpha(\{t;S : t\in\gamma(P)\})=\vee\{\alpha'(t;S) : t\in\gamma(P)\} = \vee\{sp(S)(\alpha'(t)) : t\in\gamma(P)\}$. Since $\forall S\in L$, $sp(S)$ is a complete $\vee$-morphism, $\overline{ft}(S)(P)=sp(S)(\vee\{\alpha'(t) : t\in\gamma(P)\}) = sp(S)(\alpha(\gamma(P))) = sp(S)(P)$ (theorem 5.3.0.6.(1)). Hence the best correct upper approximation of ft in A is sp.

Since $\forall S\in L$, $ft(S)$ and $sp(S)$ are complete join-morphisms, $\forall T\in T$, $\alpha(ft(S)(T)) = sp(S)(\alpha(T))$ and $\forall\phi\in A\subseteq L_t$, $\alpha(\{\phi\}) = \phi$ theorems 9.1.0.1 and 7.1.0.3.(2) imply that for all programs π, $\alpha(\mathit{lfp}(F_\pi(fp,\{\phi\}))) = \alpha(\mathrm{MOP}_\pi(fp,\{\phi\})) = \mathrm{MOP}_\pi(sp,\phi) = \mathit{lfp}(F_\pi(sp,\phi))$.

7.2.0.6.3. *Justifying the Data Flow Equations of "Available Expressions"*

Let E be the set of expressions. The set $\mathit{avail}(t)$ of expressions which are available at exit of a path $t\in T$ is defined by $\mathit{avail}(\langle\rangle)=\emptyset$ and $\forall S\in L$, $\mathit{avail}(t;S) = (\mathit{avail}(t)\cap \mathit{trans}(S))\cup \mathit{gen}(S)$ where $\mathit{trans}(S)$ is the set of expressions in E not killed by the command S while $\mathit{gen}(S)$ is the set of expressions generated by S.

An expression is available at some program point q if it is available at exit of every path from the entry point n_i to q. Therefore the set of expressions available at q is $\alpha(\mathrm{MOP}_\pi(fp,\{\lambda X.\mathit{true}\})_q)$ where $\alpha\in(2^T\to 2^E)$ is $\lambda T.\cap\{\mathit{avail}(t) : t\in T\}$.

Since α is a complete join-morphism from $2^T(\subseteq,\emptyset,T,\cup,\cap)$ onto $2^E(\supseteq,E,\emptyset,\cap,\cup)$, theorem 5.3.0.5.(3.2) defines an adjoined function γ.

According to theorem 7.2.0.4 the best correct upper approximation of ft in 2^E is $at = \lambda S\in L.[\lambda E\in 2^E.[\alpha(ft(S)(\gamma(E)))]] = \lambda S.[\lambda E.[(E\cap \mathit{trans}(S))\cup \mathit{gen}(S)]]$. Since $\forall T\in T$, $\alpha(ft(S)(T)) = at(S)(\alpha(T))$ and $\alpha(\{\lambda X.\mathit{true}\}) = \emptyset$ and $\forall S\in L$, $at(S)$ is a complete join-morphism on $2^E(\supseteq,E,\emptyset,\cap,\cup)$, theorems 9.1.0.1 and 7.1.0.3.(2) imply $\alpha(\mathrm{MOP}_\pi(fp,\{\lambda X.\mathit{true}\})) = \mathrm{MOP}_\pi(at,\emptyset) = \mathit{lfp}(F_\pi(at,\emptyset))$. Notice that $F_\pi(at,\emptyset)$ (as defined at paragraph 2 taking $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ to be $2^E(\supseteq,E,\emptyset,\cap,\cup)$) is the classical system of data flow equations for available expressions (Aho & Ullman[77]) and that the largest possible solution (least for $\supseteq$) is desired.

End of Example.

8. HIERARCHY OF PROGRAM ANALYSIS FRAMEWORKS

Once the semantics of programs has been defined by (A,τ) all program analysis frameworks (A,t,γ) are specified up to the isomorphism γ by $(\rho(A), \lambda S.[\rho\circ\tau(S)])$ where $\rho = \gamma\circ\alpha$ is an upper closure operator on A and $A \rhd\!\!\lhd\langle\alpha,\gamma\rangle A$. Program analysis frameworks can be partially ordered using the ordering of the corresponding closure operators on A since whenever $\rho_1 \sqsubseteq \rho_2$, $\rho_2(A)\subseteq\rho_1(A)$ so that program analysis frameworks corresponding to ρ_1 yield sharper information than the ones corresponding to ρ_2 (whichever global program analysis method is used).

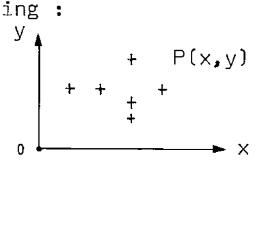
The following theorem is a constructive version of Ward[42,Th.5.3] :

THEOREM 8.0.1

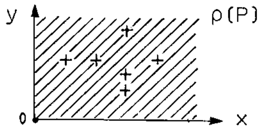
The set of upper closure operators on a complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is a complete lattice $\mathit{clo}(L\to L)(\sqsubseteq,\lambda x.x,\lambda x.\top,\lambda S.\mathit{ide}(\sqcup S),\sqcap)$.

Example 8.0.2

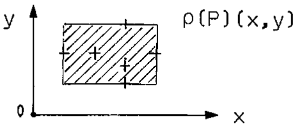
In order to briefly illustrate the hierarchy of program analysis frameworks, let us consider three comparable examples the approximation function of which can be sketched using a geometrical analogy. Let P be a predicate over two numerical variables x and y the characteristic set of which is the following :



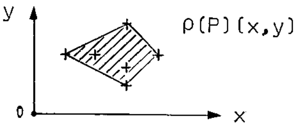
The upper closure operator of example 5.2.0.5 defines a very rough approximation consisting in approximating this set by the quarter of plane containing all its points :



A more precise approximation (example 6.3.0.5) consists in approximating the characteristic set of P by the smallest rectangle including it and whose sides run parallel with the axes :



A refinement consists in approximating the characteristic set of P by its convex-hull :



The corresponding framework was used for the automatic discovery of linear restraints among variables of programs (Cousot & Halbwachs [78]).

End of Example.

9. MERGE OVER ALL PATHS VERSUS LEAST FIXPOINT GLOBAL ANALYSIS OF PROGRAMS

9.1 "Distributive" Program Analysis Frameworks

We recalled at paragraph 4 that once a program analysis framework (A,t,γ) has been designed, the program-wide analysis problem has various solutions including the merge over all paths and least fixpoint solutions. It is known (Kam & Ullman[77]) that when A satisfies the ascending chain condition and $\forall S\in L$, $t(S)$ is isotone we have $MOP_\pi(t,\phi) \sqsubseteq lfp(F_\pi(t,\phi))$. Also the additional hypothesis that $\forall S\in L$, $t(S)$ is a join-morphism (sometimes called join-distributive map) implies $MOP_\pi(t,\phi) = lfp(F_\pi(t,\phi))$. Slightly more general is the following :

THEOREM 9.1.0.1

If $A(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is a complete lattice and $t\in(L\to(A\to A))$ is such that $\forall S\in L$, $t(S)$ is isotone then for all programs π and $\phi\in A$, $MOP_\pi(t,\phi) \sqsubseteq lfp(F_\pi(t,\phi))$. If moreover $\forall S\in L$, $t(S)$ is a complete $\sqcup$-morphism then $MOP_\pi(t,\phi) = lfp(F_\pi(t,\phi))$.

(This theorem is implicitely used at paragraph 3 taking $A = (V\to B)(\Rightarrow,\lambda X.false,\lambda X.true,\vee,\wedge)$ for $A(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ and either *sp* or *wp* for t).

If $A \rhd\langle\alpha,\gamma\rangle A$ and $t\in(L\to(A\to A))$ then the above theorem establishes the correctness of $lfp(F_\pi(t,\phi))$ with respect to $MOP_\pi(t,\phi)$. In the literature the correctness of $MOP_\pi(t,\phi)$ is generally taken for granted. Also $MOP_\pi(t,\phi)$ is considered as the desired solution to program-wide analysis problems since whenever some t(S) is not a complete join-morphism $MOP_\pi(t,\phi)$ can be strictly better than $lfp(F_\pi(t,\phi))$. When A satisfies the ascending chain condition $lfp(F_\pi(t,\phi))$ is computable, which is not necessarily the case of $MOP_\pi(t,\phi)$. In that case a variety of methods can be used (e.g. Rosen[78]) which can find sharper information that fixpoint methods and therefore approach the ideal merge over all paths solution which provides the maximum information relevant to A, t and γ.

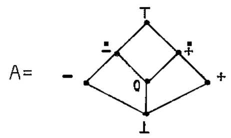
In our opinion the above argument is not entirely convincing since for different correct approximate predicate transformers $t_1, t_2\in(L\to(A\to A))$ it may be the case that $lfp(F_\pi(t_1,\phi)) \sqsubset MOP_\pi(t_2,\phi)$. In order to relieve from the burden of badly chosen approximate predicate transformers the argument must consider the best approximate predicate transformer relevant to A (theorem 7.2.0.4). Then the following result is a useful complement to theorem 9.1.0.1 :

THEOREM 9.1.0.2

Let $\bar{\tau}\in(L\to(\bar{A}\to\bar{A}))$ be the best correct upper approximation of $\tau\in(L\to(A\to A))$ in $\bar{A}=\rho(A)$. If $\rho(A)$ is a complete sublattice of A then $MOP_\pi(\bar{\tau},\phi) = lfp(F_\pi(\bar{\tau},\phi))$.

Example 9.1.0.3

If $A = (Z\to B)$ and $\bar{A}=\gamma(A)$ where :



and $\gamma(\bot)=\lambda u.false$, $\gamma(-)=\lambda u.(u<0)$, $\gamma(0)=\lambda u.(u=0)$, etc. then $\bar{A}$ is not a sublattice of A since $\gamma(-)\vee\gamma(+)\notin\gamma(A)$. The merge over all paths analysis of the program :

if x>0 *then while* x≠0 *do* x := -x; *od*; *fi*;

(which is powerful enough in order to determine that the while-loop does not terminate) is strictly better than the least fixpoint analysis (which fails to discover that $\bot$ is invariant on the exit path of the loop).

End of Example.

9.2 "Non-Distributive" Program Analysis Frameworks

The merge over all paths analysis of a program using some "non-distributive" program analysis framework can always be defined by means of the least fixpoint of a system of isotone equations associated with that program :

THEOREM 9.2.0.1

Let $A(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ be a complete lattice, $t\in(L\to(A\to A))$ be an approximate predicate transformer, $2^A(\subseteq,\emptyset,A,\cup,\cap)$ be the complete lattice of all subsets of A, $T\in(L\to(2^A\to 2^A))$ be $\lambda S.[\lambda P.\{t(S)(x):x\in P\}]$ and $\mu\in(2^A\to A)$ be $\lambda P.[\sqcup P]$.

- $\forall S\in L$, $T(S)$ is a complete $\cup$-morphism
- $\forall\pi$, $\forall\phi\in A$, $\mu(lfp(F_\pi(T,\{\phi\}))) = MOP_\pi(t,\phi)$

The above construction is not fully satisfactory since $(2^A,T)$ is not isomorphic to (A,t) when t is a complete join-morphism, so that the choice of $(2^A,T)$ in order to define $MOP_\pi(t,\phi)$ as a least fixpoint is unnecessarily too complicated. The following construction is preferable :

LEMMA 9.2.0.2

Let $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ be a complete lattice and $\overline{\mu}$, $\underline{\mu}\in(2^L\to 2^L)$ be defined as :

$\overline{\mu} = \lambda H.\mathit{if}\ H=\emptyset\ \mathit{then}\ \emptyset\ \mathit{else}\ \{\sqcup S : S\in(2^H-\emptyset)\}\ \mathit{fi}$
$\underline{\mu} = \lambda H.\mathit{if}\ H=\emptyset\ \mathit{then}\ \emptyset\ \mathit{else}\ \{\sqcap S : S\in(2^H-\emptyset)\}\ \mathit{fi}$

$\overline{\mu}$ and $\underline{\mu}$ are upper closure operators on 2^L.

Let $\sigma\in(2^L\to 2^L)$ be the join $\mathit{luis}(\overline{\mu}\cup\underline{\mu})$ of $\overline{\mu}$ and $\underline{\mu}$ in the lattice of upper closure operators on 2^L, (theorems 8.0.1 and 6.1.0.1). $\forall H\in(2^L-\emptyset)$, $\sigma(H)$ is the least complete sublattice of L containing H.

LEMMA 9.2.0.3

Let ρ be an upper closure operator on $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$. Then $\eta=\lambda x.\sqcap\{y\in\sigma(\rho(L)) : x\sqsubseteq y\}$ is the greatest closure operator on L which is less than or equal to ρ and which is a complete join-morphism.

THEOREM 9.2.0.4

(1) - Let ρ be an upper closure operator on A, $t_1\in(L\to(\overline{A}\to\overline{A}))$ be a correct upper approximation of $\tau\in(L\to(A\to A))$ in $\overline{A}=\rho(A)$. Let $\eta\in(A\to A)$ be $\lambda P.\wedge\{Q\in\sigma(\rho(A)) : P\Rightarrow Q\}$ and $t_2\in(L\to(\eta(A)\to\eta(A)))$ be $\lambda S.[\eta\circ\tau(S)]$. Then $\forall\pi$, $\forall\phi\in\overline{A}$,
$\rho(\mathit{lfp}(F_\pi(t_2,\phi))) \Rightarrow MOP_\pi(t_1,\phi)$

Moreover $\rho(\mathit{lfp}(F_\pi(t_2,\phi))) = MOP_\pi(t_1,\phi)$ whenever one of the following three conditions holds :
(2) - ρ is a complete join morphism and $t_1=\lambda S.[\rho\circ\tau(S)]$
(3) - $\forall S\in L$, $\forall P\in A$, $\rho(\tau(S)(P)) = t_1(S)(\rho(P))$
(4) - $\sigma(\rho(A))=\overline{\mu}(\rho(A))$ and $t_1=\lambda S.[\rho\circ\tau(S)]$ is the best correct upper approximation of τ in $\overline{A}=\rho(A)$.

Example 9.2.0.5

Coming back to example 9.1.0.3 where $A=(Z\to B)$ and $\rho(A)=\{\lambda u.\mathit{false},\lambda u.(u=0),\lambda u.(u<0),\lambda u.(u\leq 0),$ $\lambda u.(u>0),\lambda u.(u\geq 0),\lambda u.\mathit{true}\}$ and applying theorem 9.2.0.4 we get $\eta(A)=\rho(A)\cup\{\lambda u.(u\neq 0)\}$ so that according to 9.2.0.4.(3), $\forall\pi$, $\forall\phi\in\rho(A)$,
$\rho(\mathit{lfp}(F_\pi(\lambda S.(\eta\circ\tau(S)),\phi))) = MOP_\pi(\lambda S.(\rho\circ\tau(S)),\phi)$.

End of Example.

It is clear that when $\rho(A)$ satisfies the ascending chain condition, the construction of theorem 9.2.0.4 may lead to a refined space of approximate assertions $\eta(A)$ which does not satisfy the ascending chain condition. Then the iterative computation of $\mathit{lfp}(F_\pi(\lambda S.[\eta\circ\tau(S)],\phi))$ may not be naturally converging in a finite number of steps. Nevertheless this least fixpoint can be approximated from above using an extrapolation technique in order to accelerate the convergence of the iterates. Such a technique was developped in Cousot & Cousot[77a] using a "widening operator" $\nabla\in(\eta(A)\times\eta(A)\to\eta(A))$. In our case a possible choice of ∇ is $\lambda\langle P,Q\rangle.[\rho(P\vee Q)]$. This choice will guarantee that the refined fixpoint analysis (based on η and ∇) will be more precise than the original one (based on ρ) (but other application dependent definitions of ∇ might even be more efficient).

10. COMBINATION OF PROGRAM ANALYSIS FRAMEWORKS

The ideal method in order to construct a program analyser (to be integrated in optimizing compilers or program verification systems) would consist in a separate design and implementation of various complementary program analysis frameworks which could then be systematically combined using a once for all implemented assembler. In this section, we show that such an automatic combination of independently designed parts would not lead to an optimal analyser and that unfortunately the efficient combination of program analysis frameworks often necessitates the revision of the original design phase.

10.1 Reduced Cardinal Product of Program Analysis Frameworks

THEOREM 10.1.0.1

Let (A_1,t_1,γ_1), (A_2,t_2,γ_2) be two program analysis frameworks such that $A_1\rhd\!\!\lhd\langle,\gamma_1\rangle A$, $A_2\rhd\!\!\lhd\langle,\gamma_2\rangle A$ and t_1, t_2 are correct upper approximations of τ in A_1, A_2. The direct product (A,t,γ) of (A_1,t_1,γ_1) and (A_2,t_2,γ_2) is defined as $A=A_1\times A_2$, $t=t_1\times t_2=\lambda S.[\lambda\langle P_1,P_2\rangle.[\langle t_1(S)(P_1),t_2(S)(P_2)\rangle]]$, $\gamma=\lambda\langle P_1,P_2\rangle.(\gamma_1(P_1)\wedge\gamma_2(P_2))$.

(1) - $\forall\pi$, $\forall\phi_1\in A_1$, $\forall\phi_2\in A_2$, $MOP_\pi(t,\langle\phi_1,\phi_2\rangle) = \langle MOP_\pi(t_1,\phi_1),MOP_\pi(t_2,\phi_2)\rangle$

(2) - If moreover t_1 and t_2 are isotone, then
$\mathit{lfp}(F_\pi(t_1\times t_2,\langle\phi_1,\phi_2\rangle)) = \langle\mathit{lfp}(F_\pi(t_1,\phi_1)),\mathit{lfp}(F_\pi(t_2,\phi_2))\rangle$

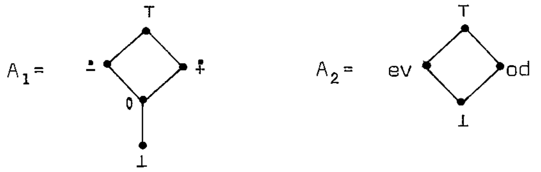
This definition of direct product is not satisfactory since γ is not necessarily injective and t is not necessarily optimal. Hence given a global program analysis algorithm we can get sharper information than the one obtained by the separate analyses just by revising the definition of A and t as stated in theorems 5.3.0.7 and 7.2.0.4 :

THEOREM 10.1.0.2

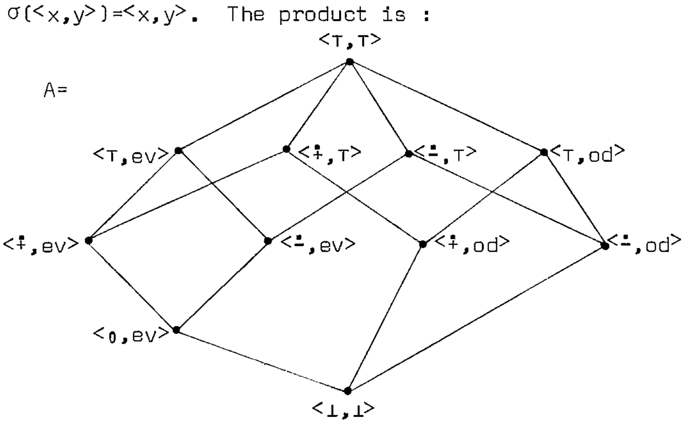
- Let (A_1,t_1,γ_1) and (A_2,t_2,γ_2) be two program analysis frameworks such that $A_1\rhd\!\!\lhd\langle\alpha_1,\gamma_1\rangle A$, $A_2\rhd\!\!\lhd\langle\alpha_2,\gamma_2\rangle A$, $t_1=\lambda S.(\alpha_1\circ\tau(S)\circ\gamma_1)$ and $t_2=\lambda S.(\alpha_2\circ\tau(S)\circ\gamma_2)$ are respectively the best upper approximation of τ in A. Let $\sigma\in((A_1\times A_2)\to(A_1\times A_2))$ be defined as $\lambda\langle P_1,P_2\rangle.\sqcap\{\langle Q_1,Q_2\rangle : \gamma_1(P_1)\wedge\gamma_2(P_2) = \gamma_1(Q_1)\wedge\gamma_2(Q_2)\}$.
- The *reduced product* $(A_1,t_1,\gamma_1)*(A_2,t_2,\gamma_2)$ is (A,t,γ) where $A=\sigma(A_1\times A_2)(\sqsubseteq,\bot,\sigma(\top),\sqcup,\lambda S.(\sigma(\sqcap S)))$, $t=\lambda S.[\alpha\circ\tau(S)\circ\gamma]$, $\gamma=\lambda\langle P_1,P_2\rangle.(\gamma_1(P_1)\wedge\gamma_2(P_2))$, $\alpha=\lambda P.(\sigma(\langle\alpha_1(P),\alpha_2(P)\rangle))$
- $A\rhd\!\!\lhd\langle\alpha,\gamma\rangle A$, $\forall S\in L$, $(\alpha\circ\tau(S)\circ\gamma)\sqsubseteq\sigma((t_1\times t_2)(S))$ and this inequality can be strict.

Since $\gamma\circ\alpha=\gamma_1\circ\alpha_1\wedge\gamma_2\circ\alpha_2$, $\sigma(A_1\times A_2)$ is a representation of the space of approximate assertions corresponding to the meet of the closure operators $\gamma_1\circ\alpha_1$ and $\gamma_2\circ\alpha_2$ (theorem 8.0.1) viz. to the join $\{\sqcap P : P\subseteq\gamma_1(A_1)\cup\gamma_2(A_2)\}$ of the Moore families $\gamma_1(A_1)$ and $\gamma_2(A_2)$.

Example 10.1.0.3



$\gamma_1(\bot)=\lambda x.\mathit{false}$, $\gamma_1(0)=\lambda x.(x=0)$, $\gamma_1(\dot{+})=\lambda x.(x\geq 0)$, $\gamma_1(\dot{-})=\lambda x.(x\leq 0)$, $\gamma_1(\top)=\lambda x.\mathit{true}$, $\gamma_2(\bot)=\lambda x.\mathit{false}$, $\gamma_2(ev)=\lambda x.(x \text{ modulo } 2 = 0)$, $\gamma_2(od)=\lambda x.(x \text{ modulo } 2 = 1)$, $\gamma_2(\top)=\lambda x.\mathit{true}$, $\forall x\in A_1$, $\forall y\in A_2$, $\sigma(\langle x,\bot\rangle)=\sigma(\langle\bot,y\rangle)=\sigma(\langle 0,od\rangle)=\langle\bot,\bot\rangle$. $\sigma(\langle 0,ev\rangle)=\sigma(\langle 0,\top\rangle)=\langle 0,ev\rangle$ otherwise $\sigma(\langle x,y\rangle)=\langle x,y\rangle$. The product is :



The following program (Manna[74 ,p.179] computes $y_3 = x_1^{x_2}$ (with the convention $0^0 = 1$) for every integer x_1 and natural number x_2 :

```
{1}   <y1,y2,y3> := <x1,x2,1>;
{2}   until y2=0 do
{3}      if odd(y2) then
{4}         <y2,y3> := <y2-1,y1*y3>;
{5}      else
{6}         <y1,y2> := <y1*y1,y2 div 2>;
{7}      fi;
{8}   od;
```

The fixpoint analysis with entry condition $\lambda(y_1,y_2,y_3,x_1,x_2).(x_2\geq 0)$ using A_1 leads to the following result for the variable y_2 :

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	$\dot{+}$	T	T	T	T	T	T	0

The fixpoint analysis using A_2 leads to the following result for the variable y_2 :

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	T	T	od	ev	ev	T	T	ev

According to theorem 10.1.0.1 the direct product of the above analyses cannot yield sharper information. On the other hand using the reduced direct product $(A-\{\langle\bot,\bot\rangle\})^5 \cup \{\langle\langle\bot,\bot\rangle,\ldots,\langle\bot,\bot\rangle\rangle\}$ and the corresponding optimal approximate predicate transformer (which takes account of the rule $\langle\dot{+},od\rangle-1 = \langle\dot{+},ev\rangle$) we get :

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	$\langle\dot{+},\top\rangle$	$\langle\dot{+},\top\rangle$	$\langle\dot{+},od\rangle$	$\langle\dot{+},ev\rangle$	$\langle\dot{+},ev\rangle$	$\langle\dot{+},\top\rangle$	$\langle\dot{+},\top\rangle$	$\langle 0,ev\rangle$

End of Example.

Remark 10.1.0.4

Let $L_1(\sqsubseteq_1)$, $L_2(\sqsubseteq_2)$ be posets. The cardinal sum of L_1 and L_2 is the set of all elements in L_1 or L_2, considered as disjoint. When $L_1(\sqsubseteq_1,\bot_1,\top_1,\sqcap_1,\sqcup_1)$ and $L_2(\sqsubseteq_2,\bot_2,\top_2,\sqcup_2,\sqcap_2)$ are complete lattices we can define the disjoint sum L_1+L_2 as $L_1\cup L_2\cup\{\bot,\top\}$ with ordering $x\sqsubseteq y$ iff $(x=\bot)$ or $(y=\top)$ or $(x,y\in L_1$, and $x\sqsubseteq_1 y)$ or $(x,y\in L_2$ and $x\sqsubseteq_2 y)$. The meaning of elements of L_1+L_2 can be defined as $\gamma(\bot)=\gamma_1(\bot_1)\wedge\gamma_2(\bot_2)$, $\gamma(x)=\gamma_1(x)$ if $x\in L_1$, $\gamma(x)=\gamma_2(x)$ if $x\in L_2$, $\gamma(\top)=\lambda x.\mathit{true}$. Even when γ_1 and γ_2 are one-to-one complete meet-morphisms, γ may be neither one-to-one nor a complete meet-morphism. In order to satisfy assumption 5.1.0.2 the set $\gamma(L_1+L_2)$ must be completed using theorem 5.2.0.4. Then it turns out that the least Moore family containing $\gamma(L_1+L_2)$ is equal to $\gamma'(L_1*L_2)$ (γ' as defined in theorem 10.1.0.2). Therefore the use of disjoint sums amounts to the use of reduced products.

End of Remark.

10.2 Reduced Cardinal Power of Program Analysis Frameworks

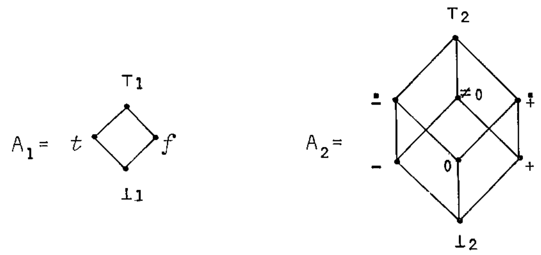
The cardinal power $L_2^{L_1}$ with base $L_2(\sqsubseteq_2,\bot_2,\top_2,\sqcup_2,\sqcap_2)$ and exponent $L_1(\sqsubseteq_1,\bot_1,\top_1,\sqcup_1,\sqcap_1)$ (hereafter noted $\mathit{iso}(L_1\rightarrow L_2)(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$) is the set of all isotone maps from L_1 to L_2 with $f\sqsubseteq g$ if and only if $f(x)\sqsubseteq_2 g(x)$ for all x in L_1. Two program analysis frameworks (A_1,t_1,γ_1) and (A_2,t_2,γ_2) can be combined by letting $g\in\mathit{iso}(L_1\rightarrow L_2)$ mean that for all x in A_1, $\gamma_2(g(x))$ holds whenever $\gamma_1(x)$ holds.

THEOREM 10.2.0.1

The *reduced cardinal power* with base (A_2,t_2,γ_2) and exponent (A_1,t_1,γ_1) is (A,t,γ) where $A=\sigma(\mathit{iso}(A_1\rightarrow A_2))$, $\sigma\in(\mathit{iso}(A_1\rightarrow A_2)\rightarrow\mathit{iso}(A_1\rightarrow A_2))$ is $\lambda g.\sqcap\{f\in\mathit{iso}(A_1\rightarrow A_2) : \gamma(f)=\gamma(g)\}$, $\gamma\in(\mathit{iso}(A_1\rightarrow A_2)\rightarrow A)$ is $\lambda g.[\lambda X.\wedge\{\gamma_1(v)(X)\Rightarrow\gamma_2(g(v))(X) : v\in A_1\}]$, $t=\lambda S.(\alpha\circ\tau(S)\circ\gamma)$ and $\alpha\in(A\rightarrow\mathit{iso}(A_1\rightarrow A_2))$ is $\lambda P.[\sigma(\lambda v.[\alpha_2(P\wedge\gamma_1(v))])]$.

$A \rhd\langle\alpha,\gamma\rangle A$ and $\forall S\in L$, $t(S)\sqsubseteq\lambda g.[\lambda v.\sqcup_2\{t_2(S)(g(z)) : z\in A_1 \wedge t_1(S)(z)\sqsubseteq v\}]$ (with $\sqcup_2\emptyset=\bot_2$).

Example 10.2.0.2



$\gamma_1(\bot_1)=\gamma_2(\bot_2)=\lambda(b,x).\mathit{false}$, $\gamma_1(\top_1)=\gamma_2(\top_2)=\lambda(b,x).\mathit{true}$, $\gamma_1(t)=\lambda(b,x).(b)$, $\gamma_1(f)=\lambda(b,x).(\neg b)$, $\gamma_2(-)=\lambda(b,x).(x<0)$, etc.

The analysis of the program :

```
{1}   x := 100; b := true;
{2}   while b do
{3}      x := x-1;
{4}      b := (x>0);
{5}   od;
```

using the reduced cardinal product of A_1 and A_2 yields no information since no relationship can be discovered between b and x.

Following theorem 10.2.0.1 we determine that if $g \in (A_1 \rightarrow A_2)$ then $\gamma(g)=(\gamma_1(t) \wedge \gamma_2(g(t))) \vee (\gamma_1(f) \wedge \gamma_2(g(f)))$. Therefore $\sigma(g)=h$ where $h(\bot_1)=\bot_2$, $h(t)=g(t)$, $h(f)=g(f)$, $h(\top_1)=g(t) \sqcup_2 g(f)$. It follows that $\sigma(\mathit{iso}(A_1 \rightarrow A_2))$ is isomorphic to $(\{t,f\} \rightarrow A_2)$ (or $A_2 \times A_2$).

The system of equations associated with the above program and the entry specification $\lambda b.\top_2$ is then :

$$\left\{ \begin{array}{lll} g_1 & = & \lambda b.\mathit{if}\ b=t\ \mathit{then}\ +\ \mathit{else}\ \bot_2\ \mathit{fi} \\ g_2 & = & \lambda b.\mathit{if}\ b=t\ \mathit{then}\ g_1(t) \sqcup_2 g_4(t)\ \mathit{else}\ \bot_2\ \mathit{fi} \\ g_3 & = & \lambda b.\mathit{decr}(g_2(b)) \\ g_4 & = & \lambda b.\mathit{if}\ b=t\ \mathit{then}\ (g_3(t) \sqcap_2 +) \sqcup_2 (g_3(f) \sqcap_2 +) \\ & & \qquad \mathit{elsif}\ b=f\ \mathit{then}\ (g_3(t) \sqcap_2 \dot{-}) \sqcup_2 (g_3(f) \sqcap_2 \dot{-})\ \mathit{fi} \\ g_5 & = & \lambda b.\mathit{if}\ b=f\ \mathit{then}\ g_1(f) \sqcup_2 g_4(f)\ \mathit{else}\ \bot_2\ \mathit{fi} \end{array} \right.$$

where $\mathit{decr}(\bot_2)=\bot_2$, $\mathit{decr}(0)=\mathit{decr}(-)=\mathit{decr}(\dot{-})=-$, $\mathit{decr}(+)=\dot{+}$, $\mathit{decr}(\neq 0)=\mathit{decr}(\dot{+})=\mathit{decr}(\top_2)=\top_2$.

The iterative resolution of this system of equations starting from the infimum $\lambda b.\bot_2$ yields $\gamma(g_1)=\gamma(g_2)=\lambda(b,x).(b \wedge x>0)$, $\gamma(g_3)=\lambda(b,x).(b \wedge x \geq 0)$, $\gamma(g_4)=\lambda(b,x).((b \wedge x>0) \vee (\neg b \wedge x=0))$, $\gamma(g_5)=\lambda(b,x).(\neg b \wedge x=0)$.

End of Example.

11. REFERENCES

Aho A.V. & Ullman J.D. [1977], *Principles of compiler design*, Addison Wesley Pub. Co., (1977).

Birkhoff G. [1967], *Lattice theory*, AMS Colloquium Pub., XXV, Third ed., Providence, R.I., (1967).

Cousot P. [1977], *Asynchronous iterative methods for solving a fixed point system of motone equations in a complete lattice*, Rapport de Recherche n°88, Laboratoire IMAG, Grenoble, (Sept. 1977).

Cousot P. & Cousot R. [1977a], *Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Conf. Rec. of the 4th ACM Symp. on Principles of Programming Languages, Los Angeles, Calif., (Jan. 1977), 238-252.

Cousot P. & Cousot R. [1977b], *Static determination of dynamic properties of recursive procedures*, IFIP WG.2.2 Working Conf. on Formal Description of Programming Concepts, St-Andrews, Canada, North-Holland Pub. Co., (Aug. 1977).

Cousot P. & Halbwachs N. [1978], *Automatic discovery of linear restraints among variables of a program*, Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages, Tucson, Ariz., (Jan. 1978), 84-97.

Dijkstra E.W. [1976], *A discipline of programming*, Prentice Hall, Englewood Cliffs, N.J., (1976).

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

Graham S.L. & Wegman M. [1976], *A fast and usually linear algorithm for global flow analysis*, JACM 23, 1(1976), 172-202.

Grätzer G. [1971], *Lattice theory, first concepts and distributive lattices*, W.H. Freeman and Co., San Francisco, Calif., (1971).

Grätzer G. & Schmidt E.T. [1958], *Ideals and congruence relations in lattices*, Acta Math. Acad. Sci. Hungar., 9(1958), 137-175.

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

Hoare C.A.R. [1978], *Some properties of predicate transformers*, JACM 25, 3(July 1978), 461-480.

Jones N.D. & Muchnick S.S. [1976], *Binding time optimization in programming languages : some thoughts toward the design of an ideal language*, Conf. Rec. of the 3rd ACM Symp. on Principles of Programming Languages, Atlanta, GA., (Jan. 1976), 77-91.

Kaplan M.A. & Ullman J.D. [1978], *A general scheme for the automatic inference of variable types*, Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages, Tucson, Ariz., (Jan. 1978), 60-75.

Kam J.B. & Ullman J.D. [1977], *Monotone data flow analysis frameworks*, Acta Informatica, 7(1977), 305-317.

Kildall G.A. [1973], *A unified approach to global program optimization*, Conf. Rec. of the ACM Symp. on Principles of Programming Languages, Boston, Mass., (Oct. 1973), 194-206.

Manna Z. [1974], *Mathematical theory of computation*, Mc-Graw Hill Book Co., (1974).

Monteiro A. & Ribeiro H. [1942], *L'opération de fermeture et ses invariants dans les systèmes partiellement ordonnés*, Portugal. Math. 3, 171-184.

Ore O. [1943], *Combination of closure relations*, Ann. of Math., 44(1943), 514-533.

Ore O. [1944], *Galois connections*, Trans. AMS, 55 (1944), 493-513.

Pickert G. [1952], *Bemerkungen über Galois-verbindungen*, Archv. Math. J. 3(1952), 285-289.

Rosen B. [1978], *Monoids for rapid data flow analysis* Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages, Tucson, Ariz.,47-59.

Scott D. [1972], *Continuous lattices*, Lect. Notes in Math. 274, Springer Verlag, 97-136.

Scott D. [1976], *Data types as lattices*, SIAM Comp. 5, 3(Sept. 1976), 522-587.

Shamir A. & Wadge W.W. [1977], *Data types as objects*, Lect. Notes in Comp. Sci. 52, Springer Verlag, (1977), 465-479.

Tarjan R.E. [1976], *Iterative algorithms for global flow analysis*, in *Algorithms and complexity, new directions and recent results*, (J.F. Traub ed.), Acad. Press Inc., (1976), 71-101.

Tenenbaum A.M. [1974], *Type determination for very high level languages*, Rep. NSO-3, Comp. Sci. Dept., N.Y. Univ., (Oct. 1974).

Ward M. [1942], *The closure operators of a lattice*, Annals Math., 43(1942), 191-196.

Wegbreit B. [1975], *Property extraction in well-founded property sets*, IEEE Trans. on Soft. Eng., SE-1, 3(Sept. 1975), 270-285.