UNIVERSITE SCIENTIFIQUE ET MEDICALE
et INSTITUT NATIONAL POLYTECHNIQUE
de GRENOBLE

MATHEMATIQUES APPLIQUEES ET INFORMATIQUE

Laboratoire associé au CNRS n°7

B.P. 53 - 38041 GRENOBLE cédex France

ANALYSIS OF THE BEHAVIOUR OF DYNAMIC DISCRETE SYSTEMS

PART I : DETERMINIST SYSTEMS

Patrick Cousot

R.R.161 Janvier 1979

RAPPORT DE RECHERCHE

ANALYSIS OF THE BEHAVIOUR OF DYNAMIC DISCRETE SYSTEMS

PART I : DETERMINIST SYSTEMS

Patrick Cousot

R.R.161 Janvier 1979

ANALYSIS OF THE BEHAVIOUR OF DYNAMIC DISCRETE SYSTEMS
PART I : DETERMINIST SYSTEMS

Patrick Cousot

Laboratoire I.M.A.G.
BP.53X, 38041 Grenoble cedex, France.

ABSTRACT :

We establish general mathematical techniques for analyzing the behaviour of dynamic discrete systems defined by a transition relation on states. The results are applied to the problem of analyzing semantic properties of programs. In this first part determinist (functional) systems and sequential programs are considered.

RESUME :

Nous établissons des techniques mathématiques générales pour analyser le comportement de systèmes dynamiques discrets définis par une relation de transition sur des états. Les résultats sont appliqués au problème de l'analyse sémantique des programmes. Dans cette première partie nous considérons le cas des systèmes déterministes (fonctionnels) et des programmes séquentiels.

ANALYSIS OF THE BEHAVIOUR OF DYNAMIC DISCRETE SYSTEMS
PART I : DETERMINIST SYSTEMS

Patrick Cousot

Laboratoire I.M.A.G.
BP.53X, 38041 Grenoble cedex, France

CONTENTS

ANALYSIS OF THE BEHAVIOUR OF DYNAMIC DISCRETE SYSTEMS
PART I : DETERMINIST SYSTEMS

Patrick Cousot

Laboratoire I.M.A.G.
BP.53X, 38041 Grenoble cedex, France

1. INTRODUCTION

We establish general mathematical techniques for analyzing the behaviour of determinist dynamic discrete systems. In order to illustrate a possible application of these results, we consider the problem of analyzing semantic properties of programs, that is, the particular case when the dynamic discrete system is defined by a sequential program.

The term "analysis of the behaviour of a determinist dynamic discrete system" will be given a precise meaning which is better introduced by the following :

Example 1.0.1

Consider the program :

```
{1}  while x≥1000 do
{2}      x:=x+y;
{3}
{4}  od;
```

where x and y are integer variables taking their values in the set I of integers included between -b-1 and b where b is the greatest machine-representable integer.

By "analysis of the semantic properties" of that program we understand the determination that:

- The execution of that program starting from the initial value $x_0 \in I$ and $y_0 \in I$ of x and y terminates without run-time error if and only if $(x_0 < 1000) \vee (y_0 < 0)$,

- The execution of that program never terminates if and only if $(1000 \le x_0 \le b) \wedge (y_0 = 0)$,
- The execution of that program leads to a run-time error (by overflow) if and only if $(x_0 \ge 1000) \wedge (y_0 > 0)$,
- During any execution of that program the following assertions P_i characterize the only possible values that the variable x and y can possess at program point i:

$$\begin{cases} P_1 = \lambda\langle x,y\rangle.[(-b-1 \le x \le b) \wedge (-b-1 \le y \le b)] \\ P_2 = \lambda\langle x,y\rangle.[(1000 \le x \le b) \wedge (-b-1 \le y \le b)] \\ P_3 = \lambda\langle x,y\rangle.[(1000+y \le x \le min(b,b+y)) \wedge (-b-1 \le y \le b)] \\ P_4 = \lambda\langle x,y\rangle.[(-b-1 \le x < 1000) \wedge (-b-1 \le y \le b)] \end{cases}$$

End of Example

2. SUMMARY

In section 3 we define what we understand by flowchart programs, that is, we define their abstract syntax and operational semantics. A program defines a dynamic discrete system (Keller[76],Pnuelli[77]) that is a transition relation on states. In section 4 we set up general mathematical techniques useful in the task of analyzing the behavior of a dynamic discrete system. In order to make this mathematically demanding section self-contained, lattice theoretical theorems on fixpoints of isotone or continuous maps are first introduced in a separate subsection. The main result of section 4 shows that the predicates characterizing the descendants of the entry states, the ascendants of the exit states, the states which lead to an error and the states which cause the system to diverge are the least or greatest solution to forward or backward fixpoint equations. This result is completed by the proof that whenever a forward equation (corresponding to post-conditions) is needed, a backward equation (corresponding to pre-conditions) can be used instead and vice versa. Finally we show that when the set of states of the dynamic discrete system is partitionned the forward or backward equation can be decomposed into a system of equations. Numerous examples of application are given which provide for a very concise presentation and justification of classical (Floyd[67],Naur[66], King[69], Hoare[69], Dijkstra[76]) or innovative program proving methods. Section 5 tailors the general mathematical techniques previously set up for analyzing the behavior of a dynamic determinist discrete system to suit the particular case when the system is a program. Two main theorems explicit the syntactic construction rules for obtaining the systems of semantic backward or forward equations from the text of a program. The facts that the extreme fixpoints of these systems of semantic equations can lead to complete information about program behavior and that the backward and forward approaches are equivalent are illustrated on the simple introductory example.

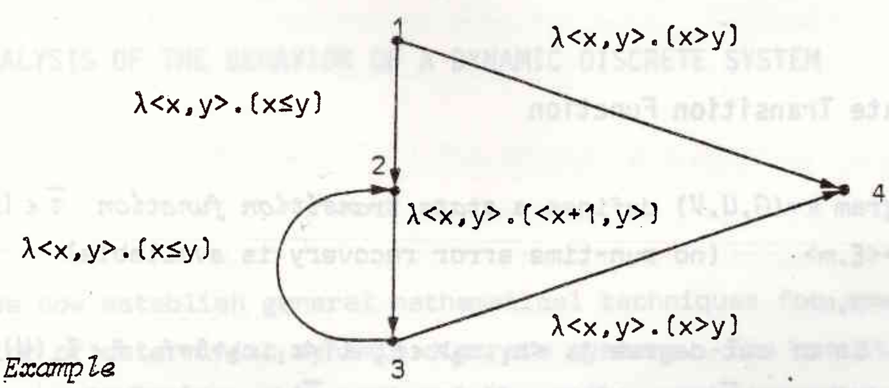
3. ABSTRACT SYNTAX AND OPERATIONAL SEMANTICS OF PROGRAMS

3.1. ABSTRACT SYNTAX

Informally programs will be abstractly represented as single-entry, single-exit directed graphs with edges labeled with instructions.

Example 3.1.0.1.

The program of example 1.0.1 will be represented as:



End of Example

A *program graph* is a triple $<V,\varepsilon,\sigma,E>$ where V is a finite set of vertices, $E \subseteq V \times V$ is a finite set of edges and $\varepsilon \in V$, $\sigma \in V$ are distinct entry and exit vertices such that ε is of in-degree 0, σ is of out-degree 0 and every vertex lies on a path from ε to σ.

Let $\vec{v}$ be a vector of variables taking their values in a *universe* U. The set $I(U)$ of *instructions* is partitionned into a subset $I_a(U)$ of *assignments* and a subset $I_t(U)$ of *tests*. An assignment $\vec{v}:=f(\vec{v})$ is represented as a partial map from U into U. A test is represented as a partial map from U into $B=\{true, false\}$.

A *program* is a triple $<G,U,L>$ where the program graph G, the universe U and the labeling $L \in (E \rightarrow I(U))$ are such that for every non-exit vertex n in G either n is of out-degree 1 and the edge leaving n is labeled with an assignment or n is of out-degree 2 and the edges leaving n are labeled with tests p and $\neg$p.

3.2. OPERATIONAL SEMANTICS

The operational semantics of a syntactically valid program π specifies the sequence of successive states of the computation defined by π.

3.2.1. States

The set S of *states* is the set of pairs $<c,m>$ where $c \in (V \cup \{\xi\})$ is the *control state* and $m \in U$ is the *memory state*. $\xi \notin V$ is the *error control state*.

The entry, exit and erroneous states are respectively characterized by $\nu_\varepsilon = \lambda<c,m>.(c=\varepsilon)$, $\nu_\sigma = \lambda<c,m>.(c=\sigma)$ and $\nu_\xi = \lambda<c,m>.(c=\xi)$.

3.2.2. State Transition Function

A program $\pi=(G,U,V)$ defines a *state transition function* $\bar{\tau} \in (S \to S)$ as follows:

- $\bar{\tau}(<\xi,m>)=<\xi,m>$ (no run-time error recovery is available)
- $\bar{\tau}(<\sigma,m>)=<\sigma,m>$
- If $c_1 \in V$ is of out-degree 1, $<c_1,c_2> \in E$, $L(<c_1,c_2>)=f$, $f \in I_a(U)$ then if $m \in dom(f)$ then $\bar{\tau}(<c_1,m>)=<c_2,f(m)>$ else $\bar{\tau}(<c_1,m>)=<\xi,m>$
- If $c_1 \in V$ is of out-degree 2, $<c_1,c_2> \in E$, $<c_1,c_3> \in E$, $L(<c_1,c_2>)=p$, $L(<c_1,c_3>)=\neg p$, $p \in I_t(U)$ then if $m \notin dom(p)$ then $\bar{\tau}(<c_1,m>)=<\xi,m>$ else if $p(m)$ then $\bar{\tau}(<c_1,m>)=<c_2,m>$ else $\bar{\tau}(<c_1,m>)=<c_3,m>$.

The *state transition relation* $\tau \in ((S \times S) \to B)$ defined by π is $\lambda<s_1,s_2>.(s_2=\bar{\tau}(s_1))$.

3.2.3. Transitive Closure of a Binary Relation

If α, $\beta \in (S \times S \to B)$ are two binary relations on S their *product* $\alpha \circ \beta$ is defined as $\lambda<s_1,s_2>.[\exists s_3 \in S : \alpha(s_1,s_3) \wedge \beta(s_3,s_2)]$. For any natural number n, the *n-extension* α^n of α is defined by recurrence as $\alpha^0 = eq = \lambda<s_1,s_2>.[s_1=s_2]$, $\alpha^{n+1}=\alpha \circ \alpha^n$. The (*reflexive*) *transitive closure* of α is $\alpha^* = \lambda<s_1,s_2>.[\exists n \geq 0 : \alpha^n(s_1,s_2)]$.

3.2.4. Execution and output of a Program

The execution of the syntactically valid program π starting from an initial state $s_1 \in S$ is said *to lead to an error* iff $[\exists s_2 \in S : \tau^*(s_1,s_2) \wedge \nu_\xi(s_2)]$, to *terminate* iff $[\exists\ s_2 \in S : \tau^*(s_1,s_2) \wedge \nu_\sigma(s_2)]$. Otherwise it is said to *diverge*. The *output* of the execution of a syntactically valid program π starting from an initial state $<\varepsilon,m_1> \in S$ is defined if and only if this execution terminates as $m_2 \in U$ such that $\tau^*(<\varepsilon,m_1>,<\sigma,m_2>)$.

4. ANALYSIS OF THE BEHAVIOR OF A DYNAMIC DISCRETE SYSTEM

We now establish general mathematical techniques for analyzing the behavior of determinist dynamic discrete systems. The results are examplified on the particular case of analyzing semantic properties of sequential programs.

4.1. DYNAMIC DISCRETE SYSTEMS

A *dynamic discrete system* is a triple $<S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi>$ such that S is a non-void set of *states*, $\tau \in ((S \times S) \to B)$ where $B = \{true, false\}$ is the *transition relation* holding between a state and its possible successors, $\nu_\varepsilon \in (S \to B)$ characterizes the *entry states*, $\nu_\sigma \in (S \to B)$ characterizes the *exit states* and $\nu_\xi \in (S \to B)$ characterizes the *erroneous states*. It is assumed that the entry, exit and erroneous states are disjoint $(\forall i,j \in \{\varepsilon,\sigma,\xi\}, (i \neq j) \Rightarrow (\forall s \in S, \neg(\nu_i(s) \wedge \nu_j(s))))$.

The following study is devoted to *total* $(\forall s_1 \in S, \exists s_2 \in S : \tau(s_1,s_2))$ and *determinist* $(\forall s_1,s_2,s_3 \in S, (\tau(s_1,s_2) \wedge \tau(s_1,s_3)) \Rightarrow (s_2=s_3))$ dynamic discrete systems.

A program as defined at paragraph 3 defines a total and determinist dynamic discrete system. Moreover the entry states are *exogenous* $(\forall s_1,s_2 \in S, \tau(s_1,s_2) \Rightarrow \neg(\nu_\varepsilon(s_2)))$, the exit states are *stable* $(\forall s_1,s_2 \in S, (\nu_\sigma(s_1) \wedge \tau(s_1,s_2)) \Rightarrow (s_1=s_2))$ and the system is *without error recovery* $(\forall s_1,s_2 \in S, (\nu_\xi(s_1) \wedge \tau(s_1,s_2)) \Rightarrow \nu_\xi(s_2))$.

The *inverse* of $\tau \in ((S \times S) \to B)$ *is* $\tau^{-1} = \lambda\langle s_1,s_2\rangle.[\tau(s_2,s_1)]$. A system is *injective* if τ^{-1} is determinist, it is *invertible* if it is injective and τ^{-1} is total. In general a program does not define an injective dynamic discrete system.

4.2. FIXPOINT THEOREMS FOR ISOTONE AND CONTINUOUS OPERATORS ON A COMPLETE LATTICE

This section recalls the lattice theoretical definitions (Birkhoff[67]) and theorems which are needed afterwards.

A *partially ordered set (poset)* $L(\sqsubseteq)$ consists of a non void set L and a binary relation $\sqsubseteq$ on L which is *reflexive* $(\forall a \in L, a \sqsubseteq a)$, *antisymmetric* $(\forall a,b \in L, (a \sqsubseteq b \wedge b \sqsubseteq a) \Rightarrow (a=b))$ and *transitive* $(\forall a,b,c \in L, (a \sqsubseteq b \wedge b \sqsubseteq c) \Rightarrow (a \sqsubseteq c))$. Given $H \subseteq L$, $a \in L$ is an *upper bound* of H if $b \sqsubseteq a$ for all $b \in H$. a is called the *least upper bound* of H, in symbols $\sqcup H$, if a is an upper bound of H and if b is any upper bound of H, then $a \sqsubseteq b$. The dualized notions (that is all $\sqsubseteq$ are replaced by the inverse $\sqsupseteq$) are the ones of *lower bound* and *greatest lower bound*. $L(\sqsubseteq)$ is a *complete lattice* if the least upper bound $\sqcup H$ of H and the greatest lower bound $\sqcap H$ of H exist for all H, $H \subseteq L$. A complete lattice L has an *infimum* $\bot = \sqcap L$ and a *supremum* $\top = \sqcup L$.

An operator f on L is *isotone* iff $(\forall a,b \in L, (a \sqsubseteq b) \Rightarrow (f(a) \sqsubseteq f(b)))$. $a \in L$ is a *fixpoint* of f iff $f(a)=a$. Tarski[55]'s fixpoint theorem states that the set of fixpoints of an isotone operator f on a complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is a (non-void) complete lattice with partial ordering $\sqsubseteq$. The *least fixpoint* of f, in symbols $lfp(f)$ is $\sqcap\{x \in L : f(x) \sqsubseteq x\}$. Dually the *greatest fixpoint* of f, in symbols *gfp*(f) is $\sqcup\{x \in L : x \sqsubseteq f(x)\}$. An element a of L such that $a \sqsubseteq f(a)$ (respectively $f(a) \sqsubseteq a$) is called a *pre-fixpoint (post-fixpoint)* of f.

Let f be an isotone operator on the complete lattice L. The *recursion induction principle* follows from Tarski's fixpoint theorem and states that $(\forall x \in L,\ (f(x) \sqsubseteq x) \Rightarrow (lfp(f) \sqsubseteq x))$. The *dual recursion induction principle* is $(\forall x \in L,\ (x \sqsubseteq f(x)) \Rightarrow (x \sqsubseteq gfp(f)))$.

If $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is a complete lattice then the set $(M \to L)$ of total maps from the set M into L is a complete lattice $(M \to L)(\sqsubseteq',\bot',\top',\sqcup',\sqcap')$ 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. The set L^n of n-tuples of elements of L is a complete lattice for the *componentwise ordering* $\langle a_1,\ldots,a_n\rangle \sqsubseteq \langle b_1,\ldots,b_n\rangle$ iff $a_i \sqsubseteq b_i$ for $i=1,\ldots,n$. The set 2^L of subsets of L is a complete lattice $2^L(\subseteq,\emptyset,L,\cup,\cap)$. A map $f \in (M \to L)$ will be extended to $(M^n \to L^n)$ as $\lambda\langle x_1,\ldots,x_n\rangle.[\langle f(x_1),\ldots,f(x_n)\rangle]$ and to $(2^M \to 2^L)$ as $\lambda S.\{f(x) : x \in S\}$.

A sequence $x_0, x_1, \ldots, x_n, \ldots$ of elements of $L(\sqsubseteq)$ is an *increasing chain* iff $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$. An operator f on $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is *semi-$\sqcup$-continuous* iff for any chain $C=\{x_i : i \in \Delta\}$, $C \subseteq L$, $f(\sqcup C)=\sqcup f(C)$. Kleene[52]'s fixpoint theorem states that the least fixpoint of a semi-$\sqcup$-continuous operator f on $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ is equal to $\sqcup\{f^i(\bot) : i \geq 0\}$ where f^i is defined by recurrence as $f^0=\lambda x.[x]$, $f^{i+1}=\lambda x.[f(f^i(x))]$.

A poset $L(\sqsubseteq)$ is said to satisfy the *ascending chain condition* if any increasing chain terminates, that is if $x_i \in L$, $i=0,1,2\ldots$, and $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$, then for some m we have $x_m=x_{m+1}=\ldots$. An operator f on $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ which is semi-$\sqcup$-continuous is necessarily isotone but the reciprocal is not true in general. However if f is an isotone operator on a complete lattice satisfying the ascending chain condition then f is semi-$\sqcup$-continuous. Also an operator f on a complete lattice L which is a *complete-$\sqcup$-morphism* (i.e. $\forall H \subseteq L,\ f(\sqcup H) = \sqcup f(H)$) is obviously semi-$\sqcup$-continuous.

Dual results hold for *decreasing chains, semi-$\sqcap$-continuous* operators, *descending chain conditions* and *complete-$\sqcap$-morphisms*.

. Suppose $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$, $L'(\sqsubseteq',\bot',\top',\sqcup',\sqcap')$ are complete lattices and we have the commuting diagram of isotone functions:

$$\begin{tikzcd} L \arrow[r, "f"] \arrow[d, "h"'] & L \arrow[d, "h"] \\ L' \arrow[r, "g"] & L' \end{tikzcd}$$

where h is strict $(h(\bot)=\bot')$ and semi-$\sqcup$-continuous. Then $h(\mathit{lfp}(f)) = \mathit{lfp}(g)$.

. In a complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$, a is a *complement* of b if $a \sqcap b=\bot$ and $a \sqcup b=\top$. A *uniquely complemented complete lattice* $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap,\neg)$ is a complete lattice in which every element a has a unique complement $\neg a$. Park[69]'s theorem states that if f is an isotone operator on a uniquely complemented complete lattice $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ then $\lambda x.[\neg f(\neg x)]$ is an isotone operator on L, $\mathit{gfp}(f) = \neg \mathit{lfp}(\lambda x.[\neg f(\neg x)])$.

. Let $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ be a complete lattice, $n\geq 1$ and F a semi-$\sqcup$-continuous operator on L^n. The system of equations:

$$X = F(X)$$

which can be detailled as:

$$\left\{ \begin{array}{l} X_j = F_j(X_1,\ldots,X_n) \\ j = 1,\ldots,n \end{array} \right.$$

has a least solution which is the least upper bound of the sequence $\{X^i : i \geq 0\}$ where $X^0=\langle\bot,\ldots,\bot\rangle$ and $X^{i+1}=F(X^i)$ which can be detailled as:

$$\left\{ \begin{array}{l} X_j^{i+1} = F_j(X_1^i,\ldots,X_n^i) \\ j = 1,\ldots,n \end{array} \right.$$

One can also use a chaotic iteration strategy and arbitrarily determine at each step which are the components of the system of equations which will evolve and in what order (as long as no component is forgotten indefinitely).

More precisely (Cousot & Cousot[77e], Cousot[77]), *lfp*(F) is the least upper bound of any chaotic iteration sequence $\{X^i : i \geq 0\}$ where $X^0 = \langle \bot, \ldots, \bot \rangle$ and

$$X_j^{i+1} = F_j(X_1^i, \ldots, X_n^i) \quad \text{if } j \in J_i$$
$$X_j^{i+1} = X_j^i \quad \text{if } j \notin J_i$$

provided that $(\forall i \geq 0,\ J_i \subseteq [1,n]$ and $(\forall j \in [1,n],\ \exists k \geq 0 : j \in J_{i+k}))$. A dual result holds for *gfp*(F).

4.3. CHARACTERIZATION OF THE SET OF DESCENDANTS OF THE ENTRY STATES OF A DYNAMIC DISCRETE SYSTEM AS A LEAST FIXPOINT

Given a discrete dynamic system $(S, \tau, \nu_\varepsilon, \nu_\sigma, \nu_\xi)$ the set of *descendants of the states satisfying a condition* $\beta \in (S \to B)$ is by definition the set characterized by :

$$\lambda s_2.[\exists s_1 \in S : \beta(s_1) \wedge \tau^*(s_1, s_2)] = post(\tau^*)(\beta)$$

using the notation

$$post \in (((S \times S) \to B) \to ((S \to B) \to (S \to B)))$$
$$post = \lambda\theta.[\lambda\beta.[\lambda s_2.[\exists s_1 \in S : \beta(s_1) \wedge \theta(s_1, s_2)]]]$$

Example 4.3.0.1.

Let π be a program defining a total and determinist system $(S, \tau, \nu_\varepsilon, \nu_\sigma, \nu_\xi)$. Assume that $\phi, \psi \in (S \to B)$ specify what it is that π is intended to do : the execution of the program π starting with an entry state satisfying ϕ terminates and the exit state satisfies Ψ on termination of π. A *partial correctness proof* consists in showing that:

$$\nu_\sigma \wedge post(\tau^*)(\nu_\varepsilon \wedge \phi) \Longrightarrow \psi$$

In words, every exit state which is descendant of an entry state satisfying ϕ must satisfy Ψ. The question of termination is not involved.

End of Example.

We now show that $post(\tau^*)(\beta)$ is a solution to the equation $\alpha = \beta \vee post(\tau)(\alpha)$, more precisely it is the least one for the implication $\Rightarrow$ considered as a partial ordering on $(S \to B)$.

THEOREM 4.3.0.2.

(1) - $((S \times S) \to B)(\Rightarrow, \lambda(s_1,s_2).false, \lambda(s_1,s_2).true, \vee, \wedge, \neg)$ and $(S \to B)(\Rightarrow, \lambda s.false, \lambda s.true, \vee, \wedge, \neg)$ are uniquely complemented complete lattices.

(2) - $\forall \theta \in ((S \times S) \to B)$, $post(\theta)$ is a strict complete $\vee$-morphism. $\forall \beta \in (S \to B)$, $\lambda\theta.[post(\theta)(\beta)]$ is a strict complete $\vee$-morphism.

(3) - $\forall \tau \in ((S \times S) \to B)$, $\forall \beta \in (S \to B)$,

$$post(\tau^*)(\beta) = \bigvee_{n \geq 0} post(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee post(\tau)(\alpha)])$$

Proof: The following diagram of isotone functions:

$$\begin{tikzcd}[column sep=huge, row sep=large]
((S \times S) \to B) \arrow[r, "\lambda\alpha.[eq \vee \alpha \circ \tau]"] \arrow[d, "\lambda\theta.[post(\theta)(\beta)]"'] & ((S \times S) \to B) \arrow[d, "\lambda\theta.[post(\theta)(\beta)]"] \\
(S \to B) \arrow[r, "\lambda\alpha.[post(eq)(\beta) \vee post(\tau)(\alpha)]"'] & (S \to B)
\end{tikzcd}$$

is commuting and $\lambda\theta.[post(\theta)(\beta)]$ is a strict complete $\vee$-morphism. Therefore $post(lfp(\lambda\alpha.[eq \vee \alpha \circ \tau]))(\beta) = post(\tau^*)(\beta) = lfp(\lambda\alpha.[post(eq)(\beta) \vee post(\tau)(\alpha)]) = lfp(\lambda\alpha.[\beta \vee post(\tau)(\alpha)])$. Also $post(\tau^*)(\beta) = post(\bigvee_{n \geq 0} \tau^n)(\beta) = \bigvee_{n \geq 0} post(\tau^n)(\beta)$.

End of Proof.

Example 4.3.0.3

Floyd[67]-Naur[66]'s method of inductive assertions for proving the partial correctness of π with respect to ϕ, Ψ, consists in guessing an assertion ι and showing that $[((\nu_\varepsilon \wedge \phi) \Rightarrow \iota) \wedge (post(\tau)(\iota) \Rightarrow \iota) \wedge ((\nu_\sigma \wedge \iota) \Rightarrow \Psi)]$.

Using the recursion induction principle, from $(((\nu_\varepsilon \wedge \phi) \Rightarrow \iota) \wedge (post(\tau)(\iota) \Rightarrow \iota))$ we infer $(lfp(\lambda\alpha.[(\nu_\varepsilon \wedge \phi) \vee post(\tau)(\alpha)]) \Rightarrow \iota)$. It follows from theorem 4.3.0.2.(3) that $(\nu_\sigma \wedge post(\tau^*)(\nu_\varepsilon \wedge \phi)) \Rightarrow (\nu_\sigma \wedge \iota) \Rightarrow \Psi$. The method is sound (Clarke[77]).

Reciprocally, if π is partially correct with respect to ϕ, Ψ then this can be proved using Floyd-Naur method. This completeness result follows from the fact that one can choose ι as $lfp(\lambda\alpha.[(\nu_\varepsilon \wedge \phi) \vee post(\tau)(\alpha)])$.

End of Example.

4.4. CHARACTERIZATION OF THE SET OF ASCENDANTS OF THE EXIT STATES OF A DETERMINIST DYNAMIC DISCRETE SYSTEM AS A LEAST FIXPOINT

In the case of a determinist dynamic discrete system, the set of *ascendants of the states satisfying a condition* $\beta \in (S \to B)$ is characterized by:

$\lambda s_1.[\exists s_2 \in S : \tau^*(s_1,s_2) \wedge \beta(s_2)] = pre(\tau^*)(\beta)$

using the notation :

$pre \in (((S \times S) \to B) \to ((S \to B) \to (S \to B)))$

$pre = \lambda\theta.[\lambda\beta.[\lambda s_1.[\exists s_2 \in S : \theta(s_1,s_2) \wedge \beta(s_2)]]]$.

Example 4.4.0.1.

Let π be a program defining a total and determinist system $(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ and ϕ, $\Psi \in (S \to B)$ be respectively an entry and exit specification. A *total correctness proof* consists in showing:

$\nu_\varepsilon \wedge \phi \Rightarrow pre(\tau^*)(\nu_\sigma \wedge \Psi)$

In words, every entry state satisfying ϕ is the ascendant of an exit state satisfying Ψ. This is a proof of termination when $\Psi = \lambda s.[true]$.

End of Example.

Once the mathematical properties of *post* have been studied similar ones can be easily derived for *pre* since $pre(\theta)(\beta) = post(\theta^{-1})(\beta)$ and $post(\theta)(\beta) = pre(\theta^{-1})(\beta)$. This point is illustrated by the proof of the following :

THEOREM 4.4.0.2.

(1) - $\forall\theta \in ((S \times S) \to B)$, $pre(\theta)$ is a strict complete $\vee$-morphism ; $\forall\beta \in (S \to B)$, $\lambda\theta.pre(\theta)(\beta)$ is a strict complete $\vee$-morphism.

(2) - $\forall \tau \in ((S \times S) \to B)$, $\forall \beta \in (S \to B)$,

$$pre(\tau^*)(\beta) = \underset{n \geq 0}{\vee} pre(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)])$$

Proof : $\forall \tau, \tau_1, \ldots \in ((S \times S) \to B)$, $(\tau_1 \circ \tau_2)^{-1} = (\tau_2^{-1} \circ \tau_1^{-1})$; $\forall n \in \mathbb{N}$, $(\tau^n)^{-1} = (\tau^{-1})^n$; $(\underset{i}{\vee}\tau_i)^{-1} = \underset{i}{\vee}(\tau_i)^{-1}$, $(\tau^*)^{-1} = (\tau^{-1})^*$. Therefore it follows from theorem 4.3.0.2 that $\forall \theta \in ((S \times S) \to B)$, $pre(\theta) = post(\theta^{-1})$ is a strict complete $\vee$-morphism. $\forall \beta \in (S \to S)$, $\lambda\theta.pre(\theta)(\beta) = \lambda\theta.post(\theta^{-1})(\beta)$ is a strict complete $\vee$-morphism. Also $pre(\tau^*)(\beta) = post((\tau^*)^{-1})(\beta) = post((\tau^{-1})^*)(\beta) = \underset{n \geq 0}{\vee} post((\tau^{-1})^n)(\beta) = \underset{n \geq 0}{\vee} post((\tau^n)^{-1})(\beta) = \underset{n \geq 0}{\vee} pre(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee post(\tau^{-1})(\alpha)]) = lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)])$. *End of Proof.*

4.5. CHARACTERIZATION OF THE STATES OF A TOTAL AND DETERMINIST SYSTEM WHICH DO NOT LEAD TO AN ERROR AS A GREATEST FIXPOINT

The entry states which are the origin of correctly terminating or diverging execution paths of a determinist program $\pi(S, \tau, \nu_\varepsilon, \nu_\sigma, \nu_\xi)$ are those which do not lead to a run-time error. They are characterized by $\nu_\varepsilon \wedge \neg pre(\tau^*)(\nu_\xi)$.

THEOREM 4.5.0.1.

Let $\tau \in ((S \times S) \to B)$ be total and determinist. $\forall \beta \in (S \to B)$,

$$\neg pre(\tau^*)(\beta) = gfp(\lambda\alpha.[\neg\beta \wedge pre(\tau)(\alpha)])$$

Proof: $\neg pre(\tau^*)(\beta) = \neg lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)]) = \neg lfp(\neg\lambda\alpha.[\neg\beta \wedge \neg pre(\tau)(\neg(\neg\alpha))])$. According to Park's fixpoint theorem this is equal to $gfp(\lambda\alpha.[\neg\beta \wedge \neg pre(\tau)(\neg\alpha)])$. Let $\overline{\tau} \in (S \to S)$ be such that $(\forall s_1, s_2 \in S, (\tau(s_1, s_2) \Longleftrightarrow (\overline{\tau}(s_1) = s_2)))$. We have $\neg pre(\tau)(\neg\alpha) = \lambda s_1.[\neg\neg\alpha(\overline{\tau}(s_1))] = \lambda s_1.[\alpha(\overline{\tau}(s_1))] = pre(\tau)(\alpha)$. *End of Proof.*

4.6. ANALYSIS OF THE BEHAVIOR OF A TOTAL DETERMINIST DYNAMIC DISCRETE SYSTEM.

Given a total and determinist system $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ we have established that the analysis of the behavior of this system can be carried out by solving fixpoint equations as follows :

THEOREM 4.6.0.1.

(1) - The set of descendants of the entry states satisfying an entry condition $\phi \in (S \to B)$ is characterized by :

$$post(\tau^*)(\nu_\varepsilon \wedge \phi) = lfp(\lambda\alpha.[(\nu_\varepsilon \wedge \phi) \vee post(\tau)(\alpha)])$$

(2) - The set of ascendants of the exit states satisfying an exit condition $\Psi \in (S \to B)$ is characterized by :

$$pre(\tau^*)(\nu_\sigma \wedge \Psi) = lfp(\lambda\alpha.[(\nu_\sigma \wedge \Psi) \vee pre(\tau)(\alpha)])$$

(3) - The set of states leading to an error is characterized by :

$$pre(\tau^*)(\nu_\xi) = lfp(\lambda\alpha.[\nu_\xi \vee pre(\tau)(\alpha)])$$

(4) - The set of states which do not lead to an error (i.e. cause the system either to properly terminate or to diverge) is characterized by :

$$\neg pre(\tau^*)(\nu_\xi) = gfp(\lambda\alpha.[\neg\nu_\xi \wedge pre(\tau)(\alpha)])$$

(5) - The set of states which cause the system to diverge is characterized by :

$$\neg pre(\tau^*)(\nu_\sigma \vee \nu_\xi) = gfp(\lambda\alpha.[\neg\nu_\sigma \wedge \neg\nu_\xi \wedge pre(\tau)(\alpha)])$$

Example 4.6.0.2

The proof that a program $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ does not terminate for the entry states satisfying a condition $\delta \in (S \to B)$ consists in proving that $\nu_\varepsilon \wedge \delta \Rightarrow \neg pre(\tau^*)(\nu_\sigma \vee \nu_\xi)$. It follows from theorem 4.6.0.1.(5) and the dual recursion induction principle that this can be done by guessing an assertion $\iota \in (S \to B)$ and proving that $\{((\nu_\varepsilon \wedge \delta) \Rightarrow \iota) \wedge (\iota \Rightarrow \neg\nu_\sigma \wedge \neg\nu_\xi \wedge pre(\tau)(\iota))\}$. *End of Example.*

4.7. RELATIONSHIPS BETWEEN *pre* AND *post*

THEOREM 4.7.0.1.

Let $\theta \in ((S{\times}S) \to \mathcal{B})$. $\forall \beta, \gamma \in (S \to \mathcal{B})$,

(1) - $pre(\theta)(\beta) = post(\theta^{-1})(\beta)$, $post(\theta)(\beta) = pre(\theta^{-1})(\beta)$

(2) - If θ is determinist then :

$post(\theta)(pre(\theta)(\beta)) = (\beta \wedge post(\theta)(true)) \Rightarrow \beta$

(3) - If θ is total then :

$\beta \Rightarrow pre(\theta)(post(\theta)(\beta))$

(4) - If θ is total and determinist then :

. $(\beta \Rightarrow pre(\theta)(\gamma))$ iff $(post(\theta)(\beta) \Rightarrow \gamma)$

. $post(\theta)(\beta) = \wedge \{\gamma \in (S \to \mathcal{B}) : \beta \Rightarrow pre(\theta)(\gamma)\}$

. $pre(\theta)(\beta) = \vee \{\gamma \in (S \to \mathcal{B}) : post(\theta)(\gamma) \Rightarrow \beta\}$

Proof: (1) $pre(\theta)(\beta) = \lambda s_1.[\exists s_2 : \theta(s_1,s_2) \wedge \beta(s_2)] = \lambda s_1.[\exists s_2 : \beta(s_2) \wedge \theta^{-1}(s_2,s_1)] = post(\theta^{-1})(\beta)$. $post(\theta)(\beta) = post((\theta^{-1})^{-1})(\beta) = pre(\theta^{-1})(\beta)$. (2) If θ is determinist then there exists $\overline{\theta} \in (S \to S)$ such that $\theta(s_1,s_2) \Leftrightarrow s_2 = \overline{\theta}(s_1)$. Therefore $post(\theta)(pre(\theta)(\beta)) = \lambda s_3.[\exists s_1 : (\exists s_2 : s_2 = \overline{\theta}(s_1) \wedge \beta(s_2)) \wedge s_3 = \overline{\theta}(s_1)] = \lambda s_3.[\exists s_1 : \beta(\overline{\theta}(s_1)) \wedge s_3 = \overline{\theta}(s_1)] = post(\theta)(true) \wedge \beta$. (3) If θ is total then $\forall s_3 \in S$, $\beta(s_3) \Rightarrow (\beta(s_3) \wedge (\exists s_2 : \theta(s_3,s_2)) \Rightarrow (\exists s_2 : \theta(s_3,s_2) \wedge (\exists s_1 : \theta(s_1,s_2) \wedge \beta(s_1))) = pre(\theta)(post(\theta)(\beta))(s_3)$. (4) If $(\beta \Rightarrow pre(\theta)(\gamma))$ then by isotony $post(\theta)(\beta) \Rightarrow post(\theta)(pre(\theta)(\gamma)) \Rightarrow \gamma$. If $(post(\theta)(\beta) \Rightarrow \gamma)$ then by isotony $\beta \Rightarrow pre(\theta)(post(\theta)(\beta)) \Rightarrow pre(\theta)(\gamma)$. $post(\theta)(\beta) = \wedge\{\gamma : post(\theta)(\beta) \Rightarrow \gamma\} = \wedge\{\gamma : \beta \Rightarrow pre(\theta)(\gamma)\}$. $pre(\theta)(\beta) = \vee\{\gamma : \gamma \Rightarrow pre(\theta)(\beta)\} = \vee\{\gamma : post(\theta)(\beta) \Rightarrow \beta\}$. *End of Proof.*

Example 4.7.0.2.

According to theorem 4.7.0.1.4, Floyd-Naur's method for proving the partial correctness of π with respect to ϕ, Ψ which consists in guessing an assertion ι and showing that $(((\nu_\varepsilon \wedge \phi) \Rightarrow \iota) \wedge (post(\tau)(\iota) \Rightarrow \iota) \wedge ((\nu_\sigma \wedge \iota) \Rightarrow \Psi))$ is equivalent to Hoare[69]'s method which consists in guessing an assertion ι and showing that $(((\nu_\varepsilon \wedge \phi) \Rightarrow \iota) \wedge (\iota \Rightarrow pre(\tau)(\iota)) \wedge ((\nu_\sigma \wedge \iota) \Rightarrow \Psi))$. *End of Example.*

We have seen that the analysis of a system consists in solving "forward" fixpoint equations of the form $\alpha = \beta \ast post(\tau)(\alpha)$ or "backward" fixpoint equations of the form $\alpha = \beta \ast pre(\tau)(\alpha)$ (where $\beta \in (S \to \mathcal{B})$ and $\ast$ is either $\vee$ or $\wedge$). In fact a forward equation is needed, a backward equation can be used instead and vice versa :

THEOREM 4.7.0.3.

$\forall \theta \in ((S \times S) \to \mathcal{B}),\ \forall \beta \in (S \to \mathcal{B}),$

$-\ post(\theta)(\beta) = \lambda \bar{s}.[\exists s_1 \in S : \beta(s_1) \wedge pre(\theta)(\lambda s.[s = \bar{s}])(s_1)]$

$-\ pre(\theta)(\beta) = \lambda \bar{s}.[\exists s_2 \in S : post(\theta)(\lambda s.[s = \bar{s}])(s_2) \wedge \beta(s_2)]$

Proof: $post(\theta)(\beta) = \lambda \bar{s}.[\exists s_1 \in S : \beta(s_1) \wedge \theta(s_1, \bar{s})] = \lambda \bar{s}.[\exists s_1 \in S : \beta(s_1) \wedge (\exists s \in S : (s = \bar{s}) \wedge \theta(s_1, s))] = \lambda \bar{s}.[\exists s_1 \in S : \beta(s_1) \wedge pre(\theta)(\lambda s.[s = \bar{s}])(s_1)]$. *End of Proof.*

Example 4.7.0.4

A total correctness proof of a program π with respect to ϕ, Ψ consists in showing that $((\nu_\varepsilon \wedge \phi) \Rightarrow pre(\tau^*)(\nu_\sigma \wedge \Psi))$ that is to say $((\nu_\varepsilon \wedge \phi) \Rightarrow lfp(\lambda \alpha.[(\nu_\sigma \wedge \Psi) \vee pre(\tau)(\alpha)]))$. Equivalently, using *post*, one can show that: $\forall \bar{s} \in S,\ (\nu_\varepsilon(\bar{s}) \wedge \phi(\bar{s})) \Rightarrow (\exists s_2 \in S : \nu_\sigma(s_2) \wedge \Psi(s_2) \wedge lfp(\lambda \alpha.[\lambda s.(s = \bar{s}) \vee post(\tau)(\alpha)])(s_2))$

More generally we have :

$post(\tau^*)(\beta) = \lambda \bar{s}.[\exists s_1 \in S : \beta(s_1) \wedge lfp(\lambda \alpha.[\lambda s.(s = \bar{s}) \vee pre(\tau)(\alpha)])(s_1)]$

$pre(\tau^*)(\beta) = \lambda \bar{s}.[\exists s_2 \in S : \beta(s_2) \wedge lfp(\lambda \alpha.[\lambda s.(s = \bar{s}) \vee post(\tau)(\alpha)])(s_2)]$

End of Example.

4.8. PARTITIONNED DISCRETE DYNAMIC SYSTEM

A dynamic discrete system $(S, \tau, \nu_\varepsilon, \nu_\sigma, \nu_\xi)$ is said to be *partitionned* if there exist $n \geq 1$, $U_1, \ldots, U_n$, $\iota_1, \ldots, \iota_n$ such that $\forall i \in [1,n]$, ι_i is a partial one to one map from S onto U_i and $\{\iota_i^{-1}(U_i) : i \in [1,n]\}$ is a partition of S, (therefore $S = \bigcup_{i=1}^{n} \iota_i^{-1}(U_i)$ and every $s \in S$ is an element of exactly one $\iota_i^{-1}(U_i)$).

When studying the behavior of a partitionned system the equations $\alpha = \beta * post(\tau)(\alpha)$ or $\alpha = \beta * pre(\tau)(\alpha)$ can be replaced by systems of equations defined as follows :

Let us define :

$\forall i \in [1,n]$, $\sigma_i \in ((S \to B) \to (U_i \to B))$, $\sigma_i = \lambda\beta.[\beta \circ \iota_i^{-1}]$, $\sigma_i^{-1} = \lambda\beta.[\lambda s.[s \in \iota_i^{-1}(U_i) \wedge \beta(\iota_i(s))]]$

$\sigma \in ((S \to B) \to (\prod_{i=1}^{n}(U_i \to B)))$, $\sigma = \lambda\beta.(\prod_{i=1}^{n}\sigma_i(\beta)) = \lambda\beta.\langle\sigma_1(\beta), \ldots, \sigma_n(\beta)\rangle$.

σ is a strict isomorphism from $(S \to B)$ onto $\prod_{i=1}^{n}(U_i \to B)$. Its inverse is

$\sigma^{-1} = \lambda\langle\beta_1, \ldots, \beta_n\rangle.[\bigvee_{i=1}^{n}\sigma_i^{-1}(\beta_i)]$.

For any isotone operator f on $(S \to B)$, the following diagram is commuting :

$$\begin{tikzcd}
(S \to B) \arrow[rr, "f"] \arrow[d, shift right, "\sigma"'] & & (S \to B) \arrow[d, shift right, "\sigma"'] \\
\prod_{i=1}^{n}(U_i \to B) \arrow[u, shift right, "\sigma^{-1}"'] \arrow[rr, "\lambda\langle\alpha_1, \ldots, \alpha_n\rangle.[\sigma \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)]"'] & & \prod_{i=1}^{n}(U_i \to B) \arrow[u, shift right, "\sigma^{-1}"']
\end{tikzcd}$$

so that the sets of pre-fixpoints, fixpoints and post-fixpoints of f coïncide (up to the isomorphism σ) with the pre-solutions, solutions and post-solutions to the *direct decomposition* of $\alpha=f(\alpha)$ on $\prod_{i=1}^{n}(U_i \to B)$ which is the system of equations:

$$\left\{\begin{array}{l}
\alpha_1 = \sigma_1 \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n) \\
\ldots \\
\alpha_n = \sigma_n \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)
\end{array}\right.$$

In particular when $f=\lambda\alpha.[\beta * post(\tau)(\alpha)]$ or $f=\lambda\alpha.[\beta * pre(\tau)(\alpha)]$ we have :

THEOREM 4.8.0.1.

$\forall i \in [1,n]$, $\sigma_i \circ \lambda\alpha.[\beta * post(\tau)(\alpha)] \circ \sigma^{-1}$ is equal to :

$$\lambda\langle\alpha_1, \ldots, \alpha_n\rangle.[\sigma_i(\beta) * (\bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(\alpha_j))]$$

whereas $\sigma_i \circ \lambda\alpha.[\beta * pre(\tau)(\alpha)] \circ \sigma^{-1}$ is equal to :

$$\lambda\langle\alpha_1,\ldots,\alpha_n\rangle.[\sigma_i(\beta) * (\bigvee_{j \in succ_\tau(i)} pre(\tau_{ij})(\alpha_j))]$$

where :

$$\tau_{ij} \in ((U_i \times U_j) \to \mathcal{B}),\ \tau_{ij} = \lambda\langle s_1,s_2\rangle.[\tau(\iota_i^{-1}(s_1),\iota_j^{-1}(s_2))]$$

$$pred_\tau = \lambda i.\{j \in [1,n] : (\exists s_1 \in U_j, \exists s_2 \in U_i : \tau_{ji}(s_1,s_2))\}$$

$$succ_\tau = \lambda i.\{j \in [1,n] : (\exists s_1 \in U_i, \exists s_2 \in U_i : \tau_{ij}(s_1,s_2))\}$$

Proof: $\sigma_i(\beta * post(\tau)(\sigma^{-1}(\alpha_1,\ldots,\alpha_n))) = \sigma_i(\beta) * \sigma_i(post(\tau)(\bigvee_{j=1}^{n} \sigma_j^{-1}(\alpha_j))) =$ $\sigma_i(\beta) * \bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1})$. Moreover $post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1} =$ $\lambda s_2.[\exists s_1 \in S : \sigma_j^{-1}(\alpha_j)(s_1) \wedge \tau(s_1,\iota_i^{-1}(s_2))] = \lambda s_2.[\exists s_1 \in U_j : \alpha_j(s_1) \wedge \tau(\iota_j^{-1}(s_1),\iota_i^{-1}(s_2))]$ $= \lambda s_2.[\exists s_1 \in U_j : \alpha_j(s_1) \wedge \tau_{ji}(s_1,s_2)] = post(\tau_{ji})(\alpha_j)$. Therefore $\bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1}) = \bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(\alpha_j)$ since $(j \notin pred_\tau(i))$ implies $\forall s_1, s_2, \neg\tau_{ji}(s_1,s_2)$. Also $pre(\tau) = post(\tau^{-1})$, $(\tau_{ij})^{-1} = \tau_{ji}$ and $succ_\tau = pred_{\tau^{-1}}$.

End of Proof.

5. SEMANTIC ANALYSIS OF PROGRAMS

The fixpoint approach to the analysis of the behavior of total determinist dynamic discrete systems is now applied to the case of programs as defined at paragraph 3.

A program $\langle G,U,L\rangle$ where $G = \langle V,\varepsilon,\sigma,E\rangle$ and $V = [1,n]-\{\xi\}$ defines a partitionned dynamic discrete system $\langle\tau,S,\nu_\varepsilon,\nu_\sigma,\nu_\xi\rangle$ where $S = ([1,n] \times U)$, $\forall i \in [1,n]$, $U_i = U$, $\iota_i = \lambda\langle c,m\rangle.m$, $\iota_i^{-1} = \lambda m.\langle i,m\rangle$. Hence two states $\langle c_1,m_1\rangle$ and $\langle c_2,m_2\rangle$ are in the same block of the partition iff $c_1 = c_2$ that is iff both states correspond to the same program point or are erroneous.

5.1. SYSTEM OF FORWARD SEMANTIC EQUATIONS ASSOCIATED WITH A PROGRAM AND AN ENTRY SPECIFICATION

The system of forward semantic equations $P=F_\pi(\phi)(P)$ associated with a program π and an entry specification $\phi \in (U \to B)$ is the direct decomposition of $\alpha=(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi)) \vee post(\tau)(\alpha)$ on $(U\to B)^n$ that is :

$$\begin{cases} P_i = \sigma_i(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi)) \vee (\underset{j \in pred_\tau(i)}{\vee} post(\tau_{ji})(P_j)) \\ i = 1,\ldots,n \end{cases}$$

From the abstract syntax and operational semantics of programs we derive a set of construction rules for obtaining this system of equations from the program text :

□ If i is the program entry point, $i=\varepsilon$ and $pred_\tau(\varepsilon)=\emptyset$ therefore $P_{\varepsilon} = \sigma_\varepsilon(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi))= \sigma_\varepsilon(\lambda<c,m>.[(c=\varepsilon) \wedge \phi(m)]) = \phi$. Otherwise $i \neq \varepsilon$ in which case $\sigma_i(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi)) = \lambda m.false$ and

$$\begin{aligned} P_i &= \underset{j \in pred_\tau(i)}{\vee} post(\tau_{ji})(P_j) \\ &= \underset{j \in pred_\tau(i)}{\vee} \lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (\overline{\tau}(<j,m_1>)=<i,m_2>)] \end{aligned}$$

when $i \neq \varepsilon$ and $i \neq \xi$ notice that $pred_\tau(i)$ is the set of the origins of the edges entering i that is the set $pred_\pi(i)$ of predecessors of the vertex i in the program graph G of π. The expression $\lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (\overline{\tau}(<j,m_1>) = <i,m_2>)]$ depends on the instruction $L(<j,i>)$ labelling the edge $<j,i>$:

□ If $<j,i>$ is labelled with an assignment $\vec{v}=f(\vec{v})$ then

$$\begin{aligned} &\lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (\overline{\tau}(<j,m_1>)=<i,m_2>)] \\ &= \lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge m_1 \in dom(f) \wedge (m_2=f(m_1))] \end{aligned}$$

□ If $<j,i>$ is labelled with a test p then :

$$\begin{aligned} &\lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (\overline{\tau}(<j,m_1>)=<i,m_2>)] \\ &= \lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (m_1 \in dom(p)) \wedge p(m_1) \wedge (m_1=m_2)] \\ &= \lambda m_2.[P_j(m_2) \wedge (m_2 \in dom(p)) \wedge p(m_2)] \end{aligned}$$

□ If $i=\xi$ then :

$$\begin{aligned} P_\xi &= \underset{j \in pred_\tau(\xi)}{\vee} \lambda m_2.[\exists m_1 \in U : P_j(m_1) \wedge (\overline{\tau}(<j,m_1>)=<\xi,m_2>)] \\ &= P_\xi \vee \underset{j \in at(\pi)}{\vee} \lambda m_2.[P_j(m_2) \wedge (m_2 \notin dom(expr(j)))] \end{aligned}$$

where $at(\pi)$ is the set of program points j preceding an assignment $\vec{v}=f(\vec{v})$ or a test $p(\vec{v})$ and $expr(j)$ is the corresponding f or p.

The above analysis can be summarized by the following :

DEFINITION 5.1.0.1.

The *system of forward semantic equations* $P=F_\pi(\phi)(P)$ *associated with a program* π *and an entry specification* $\phi \in (U \to B)$ is :

$$\begin{cases} P_\varepsilon = \phi \\ P_i = \bigvee_{j \in pred_\pi(i)} post(L(<j,i>))(P_j) & i \in ([1,n]-\{\varepsilon,\xi\}) \\ P_\xi = (\bigvee_{j \in at(\pi)} \lambda m.[P_j(m) \wedge m \notin dom(expr(j))]) \vee P_\xi \end{cases}$$

where - $\forall f \in I_a(U)$, $post(f) = \lambda P.[\lambda m.[\exists m' \in U : P(m') \wedge m' \in dom(f) \wedge m=f(m')]]$

- $\forall p \in I_t(U)$, $post(p) = \lambda P.[\lambda m.[P(m) \wedge m \in dom(p) \wedge p(m)]]$
- $at(\pi)$ is the set of program points j preceding an assignment $\vec{v}=f(\vec{v})$ or a test $p(\vec{v})$ and $expr(j)$ is the corresponding f or p.

THEOREM 5.1.0.2.

The system of forward semantic equations $P=F_\pi(\phi)(P)$ associated with a program π and an entry specification $\phi \in (U \to B)$ is the direct decomposition of $\alpha = (\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi)) \vee post(\tau)(\alpha)$ on $(U \to B)^n$.

5.2. SYSTEM OF BACKWARD SEMANTIC EQUATIONS ASSOCIATED WITH A PROGRAM AND AN EXIT SPECIFICATION

As above the abstract syntax and operational semantics of programs can be used in order to derive sets of construction rules for associating with any program π the systems of equations which are the direct decomposition of backward equations of type $\alpha = \beta * pre(\tau)(\alpha)$ on $(U \to B)^n$ that is :

$$\begin{cases} P_i = \sigma_i(\beta) \ast \underset{j \in succ_\tau(i)}{\vee} \lambda m_1.[\exists m_2 \in U : (\overline{\tau}(<m_1,i>) = <m_2,j>) \wedge P_j(m_2)] \\ i = 1,\ldots,n \end{cases}$$

The result of this study can be summarized by the following :

DEFINITION 5.2.0.1.

The *system of backward semantic equations* $P = B_\pi(\psi)(P)$ *associated with a program* π *and an exit specification* $\Psi \in (U \to B)$ is :

$$\begin{cases} P_i = \underset{j \in succ_\pi(i)}{\vee} pre(L(<i,j>))(P_j) & i \in ([1,n]-\{\sigma,\xi\}) \\ P_\sigma = \Psi \end{cases}$$

where - $\forall f \in I_a(U)$, $pre(f) = \lambda P.[\lambda m.[m \in dom(f) \wedge P(f(m))]]$

- $\forall p \in I_t(U)$, $pre(p) = \lambda P.[\lambda m.[m \in dom(f) \wedge p(m) \wedge P(m)]]$

- $succ_\pi(i)$ is the set of successors of the vertex i in the program graph of π.

THEOREM 5.2.0.2.

(1) - The direct decomposition $P=B(P)$ of $\alpha = (\nu_\sigma \wedge \sigma_\sigma^{-1}(\Psi)) \vee pre(\tau)(\alpha)$ on $(U \to B)^n$ is :

$$\begin{cases} P_i = B_\pi(\Psi)_i(P) \vee error(i) & \text{for } i \in ([1,n]-\{\xi,\sigma\}) \\ P_\sigma = \Psi \vee P_\sigma \\ P_\xi = P_\xi \end{cases}$$

where $error(i) = \lambda m \in U.[P_\xi(m) \wedge i \in at(\pi) \wedge m \notin dom(expr(i))]$

- $\forall i \in ([1,n]-\{\xi\})$, $lfp(B)_i = lfp(B_\pi(\Psi))_i$ and $lfp(B)_\xi = \lambda m.[false]$

(2) - The direct decomposition $P=B(P)$ of $\alpha = \neg\nu_\xi \wedge pre(\tau)(\alpha)$ on $(U \to B)^n$ is :

$$\begin{cases} P_i = B_\pi(\lambda m.[true])_i(P) & \text{for } i \in ([1,n]-\{\sigma,\xi\}) \\ P_\sigma = P_\sigma \\ P_\xi = \lambda m.[false] \end{cases}$$

- $\forall i \in ([1,n]-\{\xi\})$, $gfp(B)_i = gfp(B_\pi(\lambda m.[true]))_i$ and $gfp(B)_\xi = \lambda m.[false]$

(3) - The direct decomposition $P=B(P)$ of $\alpha=\nu_\xi \vee pre(\tau)(\alpha)$ on $(U \to B)^n$ is :

$$\begin{cases} P_i = B_\pi(\lambda m.[false])_i(P) \vee error(i) & \text{for } i \in ([1,n]-\{\sigma,\xi\}) \\ P_\sigma = P_\sigma \\ P_\xi = \lambda m.[true] \end{cases}$$

- The least solution to the above system of equations is equal to the least solution to :

$$\begin{cases} P_i = B_\pi(\lambda m.[false])_i(Q) \vee \lambda m.[m \notin dom(expr(i))] & \text{for } i \in ([1,n]-\{\sigma,\xi\}) \\ P_\sigma = \lambda m.[false] \\ P_\xi = \lambda m.[true] \end{cases}$$

where Q_i stands for P_i when $i \in ([1,n]-\{\sigma,\xi\})$, Q_σ stands for $\lambda m.[false]$ and Q_ξ stands for $\lambda m.[true]$

(4) - The direct decomposition of $\alpha=\neg\nu_\sigma \wedge \neg\nu_\xi \wedge pre(\tau)(\alpha)$ on $(U \to B)^n$ is :

$$\begin{cases} P_i = B_\pi(\lambda m.[false])_i(P) & \text{for } i \in ([1,n]-\{\xi\}) \\ P_\xi = \lambda m.[false] \end{cases}$$

(5) - The direct decomposition of $\alpha=\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)$ on $(U \to B)^n$ is :

$$\begin{cases} P_i = \lambda m.[\langle i,m\rangle=\bar{s}] \vee B_\pi(\lambda m.[false])_i(P) \vee error(i) & \text{for } i \in ([1,n]-\{\sigma,\xi\}) \\ P_\sigma = \lambda m.[\langle\sigma,m\rangle=\bar{s}] \vee P_\sigma \\ P_\xi = \lambda m.[\langle\xi,m\rangle=\bar{s}] \vee P_\xi \end{cases}$$

5.3. ANALYSIS OF THE BEHAVIOR OF A PROGRAM

In order to illustrate the application of theorem 4.6.0.1. to the analysis of the behavior of a program we choose the introductory example program π :

```
{1}
{2}   while x≥1000 do
{3}      x := x+y;
{4}   od;
```

It is assumed that the domain of values of the variables x and y is $\mathcal{I}=\{n \in \mathbb{Z} : -b-1 \leq n \leq b\}$ where b is the greatest and -b-1 the lowest machine representable integer.

5.3.1. Forward Semantic Analysis

The system $P=F_\pi(\phi)(P)$ (where $F_\pi(\phi) \in ((\mathcal{I}^2 \to \mathcal{B})^5 \to (\mathcal{I}^2 \to \mathcal{B})^5)$) of forward semantic equations associated with the above program π and an entry specification $\phi \in (\mathcal{I}^2 \to \mathcal{B})$ is the following:

$$\begin{cases} P_1 = \phi \\ P_2 = \lambda\langle x,y\rangle.[(P_1 \vee P_3)(x,y) \wedge (x \in \mathcal{I}) \wedge (x \geq 1000)] \\ P_3 = \lambda\langle x,y\rangle.[\exists x' \in \mathcal{I} : P_2(x',y) \wedge ((x'+y) \in \mathcal{I}) \wedge (x=x'+y)] \\ P_4 = \lambda\langle x,y\rangle.[(P_1 \vee P_3)(x,y) \wedge (x \in \mathcal{I}) \wedge (x<1000)] \\ P_\xi = \lambda\langle x,y\rangle.[((P_1 \vee P_3)(x,y) \wedge (x \notin \mathcal{I})) \vee (P_2(x,y) \wedge ((x+y) \notin \mathcal{I}))] \end{cases}$$

5.3.1.1. The *set of entry states which are ascendant of the exit states* (i.e. causes the program to terminate properly) is characterized by :

$$\begin{aligned} &\sigma_\varepsilon(\nu_\varepsilon \wedge pre(\tau^*)(\nu_\sigma)) \\ &= \sigma_\varepsilon(\lambda\bar{s}.[\exists s_2 \in S: \nu_\sigma(s_2) \wedge post(\tau^*)(\lambda s.[s=\bar{s}])(s_2)]) \quad \text{(Th.4.7.0.3)} \\ &= \lambda\bar{m}.[\exists s_2 \in S : \nu_\sigma(s_2) \wedge post(\tau^*)(\lambda s.[s=\langle\varepsilon,\bar{m}\rangle])(s_2)]) \\ &= \lambda\bar{m}.[\exists s_2 \in S : \nu_\sigma(s_2) \wedge lfp(\lambda\alpha.[(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\lambda m.[m=\bar{m}])) \vee post(\tau)(\alpha)])(s_2)] \\ &\qquad \text{(Th.4.3.0.2.(3))} \\ &= \lambda\bar{m}.[\exists s_2 \in S : \nu_\sigma(s_2) \wedge \sigma_n^{-1}(lfp(F_\pi(\lambda m.[m=\bar{m}])))(s_2)] \quad \text{(Th.5.1.0.2)} \\ &= \lambda\bar{m}.[\exists s_2 \in S : \nu_\sigma(s_2) \wedge (\bigvee_{i=1} lfp(F_\pi(\lambda m.[m=\bar{m}]))_i(\iota_i(s_2))] \\ &= \lambda\bar{m}.[\bigvee_{i=1}^{n}(\exists m_2 \in U_i : \nu_\sigma(\iota_i^{-1}(m_2)) \wedge lfp(F_\pi(\lambda m.[m=\bar{m}]))_i(m_2))] \\ &= \lambda\bar{m}.[\exists m_2 \in U_\sigma : lfp(F_\pi(\lambda m.[m=\bar{m}]))_\sigma(m_2)] \end{aligned}$$

The least fixpoint P^ω of $F_\pi(\lambda\langle x,y\rangle.[(x=\bar{x}) \wedge (y=\bar{y})])$ is computed iteratively using a chaotic iteration sequence as follows :

$P_i^0 = \lambda\langle x,y\rangle.[\mathit{false}] \quad i=1,\ldots,4,\xi$

$P_1^1 = \lambda\langle x,y\rangle.[(x=\bar{x}) \wedge (y=\bar{y})] \quad \text{where } \langle\bar{x},\bar{y}\rangle \in I^2$

$P_2^1 = \lambda\langle x,y\rangle.[(P_1^1 \vee P_3^0)(x,y) \wedge (x \in I) \wedge (x \geq 1000)]$

$\quad = \lambda\langle x,y\rangle.[(\bar{x} \in I \wedge 1000 \leq \bar{x}) \wedge (x=\bar{x}) \wedge (y=\bar{y})]$

$P_3^1 = \lambda\langle x,y\rangle.[\exists x' \in I : P_2^1(x',y) \wedge ((x'+y) \in I) \wedge (x=x'+y)]$

$\quad = \lambda\langle x,y\rangle.[(\bar{x} \in I \wedge (\bar{x}+\bar{y}) \in I \wedge 1000 \leq \bar{x}) \wedge (x=\bar{x}+\bar{y}) \wedge (y=\bar{y})]$

$P_2^2 = \lambda\langle x,y\rangle.[(P_1^1 \vee P_3^1)(x,y) \wedge (x \in I) \wedge (x \geq 1000)]$

$\quad = \lambda\langle x,y\rangle.[((\bar{x} \in I \wedge 1000 \leq \bar{x}) \wedge (x=\bar{x}) \wedge (y=\bar{y}))$

$\quad\quad \vee ((\bar{x} \in I \wedge (\bar{x}+\bar{y}) \in I \wedge 1000 \leq \bar{x} \wedge 1000 \leq (\bar{x}+\bar{y})) \wedge (x=\bar{x}+\bar{y}) \wedge (y=\bar{y}))]$

Assume as induction hypothesis that :

$P_2^k = \lambda\langle x,y\rangle.[\exists j \in [0,k-1] : \bigwedge_{i=0}^{j} ((\bar{x}+i\bar{y}) \in I \wedge 1000 \leq (\bar{x}+i\bar{y})) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]$

then :

$P_3^k = \lambda\langle x,y\rangle.[\exists x' \in I : P_2^k(x',y) \wedge ((x'+y) \in I) \wedge (x=x'+y)]$

$\quad = \lambda\langle x,y\rangle.[\exists j \in [1,k] : \bigwedge_{i=0}^{j-1} ((\bar{x}+i\bar{y}) \in I \wedge 1000 \leq (\bar{x}+i\bar{y})) \wedge ((\bar{x}+j\bar{y}) \in I) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]$

$P_2^{k+1} = \lambda\langle x,y\rangle.[(P_1^1 \vee P_3^k)(x,y) \wedge (x \in I) \wedge (x \geq 1000)]$

$\quad = \lambda\langle x,y\rangle.[\exists j \in [0,k] : \bigwedge_{i=0}^{j} ((\bar{x}+i\bar{y}) \in I \wedge 1000 \leq (\bar{x}+i\bar{y})) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]$

proving by induction on k that P_2^k is of the form assumed in the induction hypothesis. Then passing to the limit :

$P_2^\omega = \bigvee_{k \geq 0} P_2^k$

$\quad = \lambda\langle x,y\rangle.[\exists j \geq 0 : \bigwedge_{i=0}^{j} ((\bar{x}+i\bar{y}) \in I \wedge 1000 \leq (\bar{x}+i\bar{y})) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]$

$\quad = \lambda\langle x,y\rangle.[\exists j \geq 0 : (1000 \leq \mathit{min}(\bar{x},x)) \wedge (\mathit{max}(\bar{x},x) \leq b) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]$

(It is worthy to note that the use of the symbolic entry condition $\lambda\langle x,y\rangle.[(x=\bar{x}) \wedge (y=\bar{y})]$ and of the above iteration strategy corresponds to a symbolic execution of the program loop (Hantler & King[76]) with the difference that all possible execution paths are considered simultaneously and the induction step as well as the passage to the limit deal with infinite paths). The remaining components of $\bar{lfp}(F_\pi(\lambda\langle x,y\rangle.[(x=\bar{x}) \wedge (y=\bar{y})])$ are :

$$P_1^\omega = \lambda\langle x,y\rangle.[(x=\bar{x}) \wedge (y=\bar{y})]$$

$$\begin{aligned}P_3^\omega &= \lambda\langle x,y\rangle.[\exists x' \in I : P_2^\omega(x',y) \wedge ((x'+y) \in I) \wedge (x=x'+y)] \\ &= \lambda\langle x,y\rangle.[\exists j\geq 1 : (1000 \leq \mathit{min}(\bar{x},x-\bar{y})) \wedge (\mathit{max}(\bar{x},x)\leq b) \wedge (x=\bar{x}+j\bar{y}) \wedge (y=\bar{y})]\end{aligned}$$

$$\begin{aligned}P_4^\omega &= \lambda\langle x,y\rangle.[(P_1^\omega \vee P_3^\omega)(x,y) \wedge (x \in I) \wedge (x<1000)] \\ &= \lambda\langle x,y\rangle.[((\bar{x}<1000) \wedge (x=\bar{x}) \wedge (x=\bar{y})) \\ &\qquad \vee((\bar{x}\geq 1000) \wedge (\bar{y}<0) \wedge (x=\bar{x}+(((\bar{x}-1000)\,\mathit{div}\,|\bar{y}|)+1)\bar{y}) \wedge (y=\bar{y}))]\end{aligned}$$

$$\begin{aligned}P_\xi^\omega &= \lambda\langle x,y\rangle.[((P_1^\omega \vee P_3^\omega)(x,y) \wedge (x \notin I)) \vee (P_2^\omega(x,y) \wedge ((x+y) \notin I))] \vee P_\xi^0 \\ &= \lambda\langle x,y\rangle.[(\bar{x}\geq 1000) \wedge (\bar{y}>0) \wedge (x=\bar{x}+((b-\bar{x})\,\mathit{div}\,\bar{y})\bar{y}) \wedge (y=\bar{y})]\end{aligned}$$

The set of entry states which causes the program to properly terminate is characterized by :

$$\begin{aligned}&\lambda\langle\bar{x},\bar{y}\rangle.[\exists x_2,y_2 \in I : P_4^\omega(x_2,y_2)] \\ &= \lambda\langle x,y\rangle.[(\bar{x}<1000) \vee (\bar{y}<0)]\end{aligned}$$

5.3.1.2. The *set of entry states leading to a run-time error* is characterized by :

$$\begin{aligned}&\sigma_\varepsilon(\nu_\varepsilon \wedge \mathit{pre}(\tau^*)(\nu_\xi)) \\ &= \lambda\bar{m}.[\exists m_2 \in \mathcal{U}_\xi : \mathit{lfp}(F_\pi(\lambda m.[m=\bar{m}]))_\xi(m_2)]\end{aligned}$$

that is :

$$\begin{aligned}&\lambda\langle\bar{x},\bar{y}\rangle.[\exists x_2,y_2 \in I : P_\xi^\omega(x_2,y_2)] \\ &= \lambda\langle\bar{x},\bar{y}\rangle.[(\bar{x}\geq 1000) \wedge (\bar{y}>0)]\end{aligned}$$

5.3.1.3. The *set of entry states which cause the program to diverge* is characterized by :

$$\begin{aligned}&\sigma_\varepsilon(\nu_\varepsilon \wedge \neg\mathit{pre}(\tau^*)(\nu_\sigma \vee \nu_\xi)) \\ &= \lambda\bar{m}.\neg[\bigvee_{i=1}^{n}(\exists m_2 \in \mathcal{U}_i : (\nu_\sigma \vee \nu_\xi)(\langle i,m_2\rangle) \wedge \mathit{lfp}(F_\pi(\lambda m.[m=\bar{m}]))_i(m_2))] \\ &= \lambda\bar{m}.[\ \neg(\exists m_2 \in \mathcal{U}_\sigma : \mathit{lfp}(F_\pi(\lambda m.[m=\bar{m}]))_\sigma(m_2)) \\ &\qquad \wedge \neg(\exists m_2 \in \mathcal{U}_\xi : \mathit{lfp}(F_\pi(\lambda m.[m=\bar{m}]))_\xi(m_2))]\end{aligned}$$

that is :

$$\lambda<\overline{x},\overline{y}>.[\neg(\exists x_2,y_2 \in I : P_4^{\omega}(x_2,y_2)) \wedge \neg(\exists x_2,y_2 \in I : P_{\xi}^{\omega}(x_2,y_2))]$$
$$= \lambda<\overline{x},\overline{y}>.[(\overline{x}\geq 1000) \wedge (y=0)]$$

5.3.1.4. The *set of descendants of the entry states satisfying the entry condition* $\phi \in (I^2 \to B)$ is characterized by $post(\tau^*)(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi))$ that is (Th.4.6.0.1 and Th.5.1.0.2) up to the isomorphism σ by $Q^{\omega}=lfp(F_\pi(\phi))$:

$$\begin{cases} Q_1^{\omega} = \phi \\ Q_2^{\omega} = \lambda<x,y>.[\exists j\geq 0 : \phi(x-jy,y) \wedge (1000\leq min(x-jy,x)) \wedge (max(x-jy,x)\leq b)] \\ Q_3^{\omega} = \lambda<x,y>.[\exists j\geq 1 : \phi(x-jy,y) \wedge (1000\leq min(x-jy,x-y)) \wedge (max(x-jy,x)\leq b)] \\ Q_4^{\omega} = \lambda<x,y>.[(\phi(x,y) \wedge (x<1000)) \vee ((y<0) \wedge (\exists j\geq 1 : \phi(x-jy,y) \wedge (x-jy\leq b) \wedge \\ \qquad (x<1000\leq x-y))] \\ Q_{\xi}^{\omega} = \lambda<x,y>.[(y>0)\wedge (\exists j\geq 0 : \phi(x-jy,y) \wedge (1000\leq x-jy<x\leq b<x+y))] \end{cases}$$

Equivalently Q^{ω} can be obtained from P^{ω} as follows :

$$\sigma_i(post(\tau^*)(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi))$$
$$= \sigma_i(\lambda s_2.[\exists\overline{s} : \nu_\varepsilon(\overline{s}) \wedge \sigma_\varepsilon^{-1}(\phi)(\overline{s})\wedge post(\tau^*)(\lambda s.[s=\overline{s}])(s_2)])$$
$$= \lambda m_2.[\exists\overline{m} : \phi(\overline{m}) \wedge post(\tau^*)(\lambda s.[s=<\varepsilon,\overline{m}>])(<i,m_2>)]$$
$$= \lambda m_2.[\exists\overline{m} : \phi(\overline{m})\wedge \sigma^{-1}(lfp(F_\pi(\lambda m.[m=\overline{m}])))(<i,m_2>)]$$
$$= \lambda m_2.[\exists\overline{m} : \phi(\overline{m})\wedge lfp(F_\pi(\lambda m.[m=\overline{m}]))_i(m_2)]$$

therefore at each program point i the set of descendants of the entry states satisfying the entry condition $\phi \in (I^2 \to B)$ is characterized by :

$$Q_i^{\omega} = \lambda<x,y>.[\exists\overline{x},\overline{y} \in I^2 : \phi(\overline{x},\overline{y}) \wedge P_i^{\omega}(x,y)]$$

For example :

$$Q_{\xi}^{\omega} = \lambda(x,y).[\exists\overline{x},\overline{y} \in I^2 : \phi(\overline{x},\overline{y}) \wedge (\overline{x}\geq 1000) \wedge (\overline{y}>0)\wedge (x=\overline{x}+((b-\overline{x})div\ \overline{y})\overline{y}) \wedge (y=\overline{y})]$$
$$= \lambda(x,y).[\exists\overline{x}\in I : (\exists j : \phi(x-jy,y)\wedge (x-jy\geq 1000) \wedge (y>0) \wedge (x=\overline{x}+jy)\wedge j=(b-\overline{x})div\ y)]$$
$$= \lambda(x,y).[(y>0) \wedge (\exists j\geq 0 : \phi(x-jy,y)\wedge (1000\leq x-jy<x\leq b) \wedge (j=j+(b-x)div\ y))]$$
$$= \lambda(x,y).[(y>0) \wedge (\exists j\geq 0 : \phi(x-jy,y)\wedge (1000\leq x-jy<x\leq b<x+y))]$$

We now recommence the semantic analysis of this program but this time using backward equations.

5.3.2. Backward Semantic Analysis

The system $P=B_\pi(\Psi)(P)$ (where $B_\pi(\Psi)\in((I^2\to B)^4\to(I^2\to B)^4)$) of backward semantic equations associated with the example program π and an exit specification $\Psi\in(I^2\to B)$ is the following :

$$\begin{cases} P_1 = \lambda<x,y>.[((x\in I)\wedge(x\geq 1000)\wedge P_2(x,y))\vee((x\in I)\wedge(x<1000)\wedge P_4(x,y))] \\ P_2 = \lambda<x,y>.[((x+y)\in I)\wedge P_3(x+y,y)] \\ P_3 = \lambda<x,y>.[((x\in I)\wedge(x\geq 1000)\wedge P_2(x,y))\vee((x\in I)\wedge(x<1000)\wedge P_4(x,y))] \\ P_4 = \Psi \end{cases}$$

5.3.2.1. The *set of entry states which are ascendant of the exit states* (i.e. cause the program to terminate properly) is characterized by :

$$\begin{aligned}
&\sigma_\varepsilon(\nu_\varepsilon\wedge pre(\tau^*)(\nu_\sigma)) \\
&= \sigma_\varepsilon(\nu_\varepsilon\wedge lfp(\lambda\alpha.[\nu_\sigma\vee pre(\tau)(\alpha)])) \qquad (\text{Th.4.6.0.1.(2)}) \\
&= \sigma_\varepsilon(\nu_\varepsilon\wedge\sigma^{-1}(lfp(B_\pi(\lambda m.[true])))) \qquad (\text{Th.5.2.0.2.(1)}) \\
&= \sigma_\varepsilon(\bigvee_{i\in[1,n]}\lambda s.[\nu_\varepsilon(s)\wedge s\in\iota_i^{-1}(U)\wedge lfp(B_\pi(\lambda m.[true]))_i(\iota_i(s))]) \\
&= \sigma_\varepsilon(lfp(B_\pi(\lambda m.[true]))_\varepsilon\circ\iota_\varepsilon) \\
&= lfp(B_\pi(\lambda m.[true]))_\varepsilon
\end{aligned}$$

The least fixpoint P^ω of the above system of equations where $\Psi=\lambda<x,y>.[true]$ is :

$$\begin{cases} P_1^\omega = \lambda<x,y>.[(x<1000)\vee(y<0)] \\ P_2^\omega = \lambda<x,y>.[((x+y)\in I)\wedge((x+y<1000)\vee(y<0))] \\ P_3^\omega = \lambda<x,y>.[(x<1000)\vee(y<0)] \\ P_4^\omega = \lambda<x,y>.[true] \end{cases}$$

5.3.2.2. The *set of entry states which do not lead to a run-time error* (i.e. cause the program to properly terminate or diverge) is characterized by :

$$\begin{aligned}
&\sigma_\varepsilon(\nu_\varepsilon\wedge\neg pre(\tau^*)(\nu_\xi)) \\
&= \sigma_\varepsilon(\nu_\varepsilon\wedge gfp(\lambda\alpha.[\neg\nu_\xi\wedge pre(\tau)(\alpha)])) \qquad (\text{Th.4.6.0.1.(3)}) \\
&= \sigma_\varepsilon(\nu_\varepsilon\wedge\sigma^{-1}(gfp(B_\pi(\lambda m.[true])))) \qquad (\text{Th.5.2.0.2.(2)}) \\
&= gfp(B_\pi(\lambda m.[true]))_\varepsilon
\end{aligned}$$

The greatest fixpoint Q^ω of the above system of equations where $\Psi=\lambda\langle x,y\rangle.[true]$ can be computed iteratively starting from $Q_i^0=\lambda\langle x,y\rangle.[true]$, i=1..4, inventing the general term of a chaotic iteration sequence and passing to the limit :

$$\begin{cases} Q_1^\omega = \lambda\langle x,y\rangle.[(y\le 0) \vee (x<1000)] \\ Q_2^\omega = \lambda\langle x,y\rangle.[((x+y\le 0) \vee (x+y<1000)) \wedge ((x+y) \in I)] \\ Q_3^\omega = \lambda\langle x,y\rangle.[(y\le 0) \vee (x<1000)] \\ Q_4^\omega = \lambda\langle x,y\rangle.[true] \end{cases}$$

5.3.2.3. The *set of entry states leading to a run-time error* is characterized by $\lambda\langle x,y\rangle \in I^2.[\neg Q_1^\omega(x,y)] = \lambda\langle x,y\rangle \in I^2.[(y>0) \wedge (x\ge 1000)]$.

Equivalently the set of ascendants of the run-time error states is characterized by $pre(\tau^*)(\nu_\xi)$ which according to Th.4.6.0.1.(3) and Th.5.2.0.2.(3) is equal (up to the isomorphism σ) to the least solution R^ω to :

$$\begin{cases} P_1 = \lambda\langle x,y\rangle.[(x\ge 1000) \wedge P_2(x,y)] \\ P_2 = \lambda\langle x,y\rangle.[(((x+y) \in I) \wedge P_3(x+y,y)) \vee ((x+y) \notin I)] \\ P_3 = \lambda\langle x,y\rangle.[(x\ge 1000) \wedge P_2(x,y)] \\ P_4 = \lambda\langle x,y\rangle.[false] \\ P_\xi = \lambda\langle x,y\rangle.[true] \end{cases}$$

that is $R_1^\omega = R_3^\omega = \lambda\langle x,y\rangle.[(x\ge 1000) \wedge (y>0)]$, $R_2^\omega = \lambda\langle x,y\rangle.[((x+y)\ge 0 \wedge (y>0)) \vee ((x+y) \notin I)]$, $R_4^\omega = \lambda\langle x,y\rangle.[false]$, $R_\xi^\omega = \lambda\langle x,y\rangle.[true]$.

5.3.2.4. The *set of entry states which cause the program to diverge* is characterized by $\lambda\langle x,y\rangle.[Q_1^\omega(x,y) \wedge \neg P_1^\omega(x,y)] = \lambda\langle x,y\rangle.[(x\ge 1000) \wedge (y=0)]$

Equivalently the states which cause the program to diverge can be characterized by $\neg pre(\tau^*)(\nu_\xi \vee \nu_\sigma) = gfp(\lambda\alpha.[\neg\nu_\sigma \wedge \neg\nu_\xi \wedge pre(\tau)(\alpha)])$ which according to Theorem 5.2.0.2.(4) is equal (up to the isomorphism σ) to $D^\omega = gfp(B_\pi(\lambda\langle x,y\rangle.[false]))$ that is $D_1^\omega = D_2^\omega = D_3^\omega = \lambda\langle x,y\rangle.[(x\ge 1000) \wedge (y=0)]$ and $D_4^\omega = D_\xi^\omega = \lambda\langle x,y\rangle.[false]$.

5.3.2.5. The *set of descendants of the input states satisfying an entry condition* $\phi \in (I^2 \to B)$ is characterized by :

$$
\begin{aligned}
& post(\tau^*)(\nu_\varepsilon \wedge \sigma_\varepsilon^{-1}(\phi)) \\
&= \lambda\bar{s}.[\exists s_1 \in S : \nu_\varepsilon(s_1) \wedge \sigma_{\varepsilon_1}^{-1}(\phi)(s_1) \wedge lfp(\lambda\alpha.[\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)])(s_1)] \\
&= \lambda\bar{s}.[\exists m_\varepsilon \in U_\varepsilon : \phi(m_\varepsilon) \wedge \sigma^{-1}(lfp(\sigma \circ \lambda\alpha.[\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)] \circ \sigma^{-1}))(<\varepsilon, m_\varepsilon>)] \\
&= \lambda\bar{s}.[\exists m_\varepsilon \in U_\varepsilon : \phi(m_\varepsilon) \wedge lfp(\sigma \circ \lambda\alpha.[\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)] \circ \sigma^{-1})_\varepsilon(m_\varepsilon)]
\end{aligned}
$$

According to theorem 5.2.0.2.(5) the direct decomposition of $\lambda\alpha.[\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)]$ is the following when τ is defined by our example program :

$$
\begin{cases}
P_1 = \lambda<x,y>.[(<1,<x,y>>=\bar{s}) \vee (x \in I \wedge x \geq 1000 \wedge P_2(x,y)) \vee (x \in I \wedge x \leq 1000 \wedge P_4(x,y)) \\
\qquad \vee (P_\xi(x,y) \wedge x \notin I)] \\
P_2 = \lambda<x,y>.[(<2,<x,y>>=\bar{s}) \vee (x+y \in I \wedge P_3(x+y,y)) \vee (P_\xi(x,y) \wedge x+y \notin I)] \\
P_3 = \lambda<x,y>.[(<3,<x,y>>=\bar{s}) \vee (x \in I \wedge x \geq 1000 \wedge P_2(x,y)) \vee (x \in I \wedge x<1000 \wedge P_4(x, \\
\qquad \vee (P_\xi(x,y) \wedge x \notin I)] \\
P_4 = \lambda<x,y>.[(<4,<x,y>>=\bar{s}) \vee P_4(x,y)] \\
P_\xi = \lambda<x,y>.[(<\xi,<x,y>>=\bar{s}) \vee P_\xi(x,y)]
\end{cases}
$$

If $P^\omega(\bar{s})$ denotes $lfp(\sigma \circ \lambda\alpha.[\lambda s.[s=\bar{s}] \vee pre(\tau)(\alpha)] \circ \sigma^{-1})$ we determine that:

$$
\begin{aligned}
P_1^\omega(\bar{s}) = \lambda<x,y>.[&(<1,<x,y>>=\bar{s}) \\
&\vee(\exists j \geq 0 : (\forall i \in [0,j], 1000 \leq x+iy \leq b) \wedge (<2,<x+jy,y>>=\bar{s})) \\
&\vee(\exists j \geq 1 : (\forall i \in [0,j-1], 1000 \leq x+iy \leq b) \wedge (x+jy \in I) \wedge (<3,<x+jy,y>>=\bar{s})) \\
&\vee(\exists j \geq 0 : (\forall i \in [0,j-1], 1000 \leq x+iy \leq b) \wedge (x+jy \in I) \wedge (x+jy<1000) \\
&\quad \wedge (<4,<x+jy,y>>=\bar{s})) \\
&\vee(\exists j \geq 1 : (\forall i \in [0,j-1], 1000 \leq x+jy \leq b) \wedge (x+jy \notin I) \wedge (<\xi,<x+(j-1)y,y>>=\bar{s}
\end{aligned}
$$

At program point i, the set of descendants of the input states satisfying ϕ is

$$
\begin{aligned}
&\sigma_i(\lambda\bar{s}.[\exists m_\varepsilon \in U_\varepsilon : \phi(m_\varepsilon) \wedge P_\varepsilon^\omega(\bar{s})(m_\varepsilon)] \\
&= \lambda m.[\exists m_\varepsilon \in U_\varepsilon : \phi(m_\varepsilon) \wedge P_\varepsilon^\omega(<i,m>)(m_\varepsilon)]
\end{aligned}
$$

For our example:

$$
\lambda<x,y>.[\exists<x_\varepsilon,y_\varepsilon> \in I^2 : \phi(x_\varepsilon,y_\varepsilon) \wedge P_1^\omega(<i,<x,y>>)(<x_\varepsilon,y_\varepsilon>)]
$$

when i=2 this is equal to :

$$
\begin{aligned}
&\lambda<x,y>.[\exists<x_\varepsilon,y_\varepsilon> \in I^2 : \phi(x_\varepsilon,y_\varepsilon) \wedge (\exists j \geq 0 : \forall i \in [0,j], 1000 \leq x_\varepsilon+iy_\varepsilon \leq b) \\
&\qquad \wedge (<2,<x_\varepsilon+jy_\varepsilon,y_\varepsilon>>=<2,<x,y>>)]] \\
&= \lambda<x,y>.[\exists j \geq 0 : \phi(x-jy,y) \wedge 1000 \leq min(x-jy,x) \wedge (max(x-jy,x) \leq b)]
\end{aligned}
$$

5.3.3. FORWARD VERSUS BACKWARD SEMANTIC ANALYSIS OF PROGRAMS

In the literature on program verification, backward program analysis is often preferred above forward analysis (e.g. Dijkstra[76]). Theorem 4.6.0.1(1)-(2) clearly shows that the two approaches are not strictly equivalent but this point of view must be completed by theorem 4.7.0.3. and paragraph 5.3 which show that using symbolic variables one approach can serve as a substitute for the other.

6. CONCLUSION

We have established general mathematical techniques for analyzing the behavior of dynamic discrete systems defined by a transition relation on states. In this first part total and determinist systems have been considered. The study is also applicable to partial determinist systems which up to an isomorphism are equivalent to total systems with some additional undefined state. The more complicated case of non-determinist dynamic discrete systems is studied in a forthcoming second part.

The methods for analyzing the behavior of determinist dynamic discrete systems have been applied to the problem of analyzing semantic properties of sequential programs (but the applications are not necessarily confined with computer science). The advantage of using the model of discrete dynamic systems for studying program analysis methods is that the reasoning on a set of states and a state transition relation and fixpoints of isotone equations leads to very concise notations, terse results and brief proofs.

ACKNOWLEDGEMENTS

I owe a deep dept to C. PAIR whose argumented advice to study program analysis techniques using the model of dynamic discrete systems was very helpful. I wish to thank R. COUSOT for her collaboration.

7. REFERENCES

BIRKHOFF G. [1967], *Lattice theory*, AMS Colloquium Publications, XXV, Third edition, Providence, R.I., U.S.A., (1967).

CLARKE E.M. Jr. [1977], *Program invariants as fixed points*, Proc. 18th Annual Symp. on Foundations of Computer Science, Providence, R.I., U.S.A., (Oct.31 - Nov.2 1977), 18-29.

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

COUSOT P. [1978], *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*, Thèse d'état ès sciences mathématiques, Univ. of Grenoble I, (March 1978).

COUSOT P. & COUSOT R. [1977], *Automatic synthesis of optimal invariant assertions: mathematical foundations*, Proc of ACM Symp. on Artificial Intelligence & Programming Languages, Rochester, N.Y., SIGPLAN Notices 12, 8 (Aug. 1977), 1-12.

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

HANTLER S.L. & KING J.C. [1976], *An introduction to proving the correctness of programs*, Comp. Surveys 8, 3(Sept. 1976), 331-353.

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

KELLER R.M. [1976], *Formal verification of parallel programs*, CACM 19, 7(July 1976), 371-384.

KING J. [1969], *A program verifier*, Ph.D. Th., Dept. of Comp. Sci., Carnegie Mellon U., Pittsburg, Pa., U.S.A., (1969).

KLEENE S.C. [1952], *Introduction to metamathematics*, North-Holland Pub. Co., Amsterdam, (1952).

NAUR P. [1966], *Proof of algorithms by general snapshots*, BIT 5, (1966), 310-316.

PARK D. [1969], *Fixpoint induction and proofs of program properties*, Machine Intelligence 5 (1969), 59-78.

PNUELI A. [1977], *The temporal logic of programs*, Proc. 18th Annual Symp. on Foundations of Comp. Sci., Providence, R.I., (Oct.31 - Nov.2 1977), 46-57.

TARSKI A. [1955], *A lattice theoretical fixpoint theorem and its applications*, Pacific J. Math., 5(1955), 285-310.