AFCS / Spring 2014

# *Advanced Topics in the Theory of Partially Ordered Sets*

Prof.Dr. Ferucio Laurenţiu Ţiplea

"Al.I.Cuza" University of Iaşi
Department of Computer Science
Iaşi, Romania
E-mail: fltiplea@mail.dntis.ro

# Contents

# 1. Motivation

Complete partially ordered sets are crucial structures for defining the denotational semantics of programming languages and for performing static analysis.

Three well-known semantics that can be associated to a programming language:

1. operational – the meaning of a construct is specified by the computation it induces when it is executed on a machine.
2. denotational – functions are associated to each atomic construct, and composition of functions to more complex constructs.
3. axiomatic – specific properties of executing the constructs are expressed as assertions.

# 1. Motivation

A simple imperative programming language (**`while`**)

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \\
b & ::= & \mathtt{true} \mid \mathtt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & x := a \mid \mathtt{skip} \mid S_1 ; S_2 \mid \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2 \mid \mathtt{while}\ b\ \mathtt{do}\ S
\end{array}
$$

where

- $n$ ranges over numerals, $Num$;
- $x$ ranges over variables, $Var$;
- $a$ ranges over arithmetic expressions, $Aexp$;
- $b$ ranges over boolean expressions, $Bexp$;
- $S$ ranges over statements, $Stmt$.

Denotational semantics:

- State: associates to each variable its current value

$$s : Var \rightarrow \mathbf{Z}$$

The set of all states, denoted by $State$, is the set of all functions from $Var$ into $\mathbf{Z}$

$$(Var \rightarrow \mathbf{Z})$$

These functions (states) are total functions.

Denotational semantics:

- Meaning of arithmetic expressions:

$$\mathcal{A} : Aexp \to (State \to \mathbf{Z})$$

  - $\mathcal{A}(n)(s)$ is the natural number $\mathbf{n}$;
  - $\mathcal{A}(x)(s) = s(x)$;
  - $\mathcal{A}(a_1 + a_2)(s) = \mathcal{A}(a_1)(s) + \mathcal{A}(a_2)(s)$;
  - $\mathcal{A}(a_1 * a_2)(s) = \mathcal{A}(a_1)(s) * \mathcal{A}(a_2)(s)$;
  - $\mathcal{A}(a_1 - a_2)(s) = \mathcal{A}(a_1)(s) - \mathcal{A}(a_2)(s)$.

"$+$" in the left hand side is a symbol, but "$+$" in the right hand side is the usual addition operation on integers (and similar for "$*$" and "$-$").

Denotational semantics:

- Meaning of Boolean expressions:

$$\mathcal{B} : Bexp \to (State \to \mathbf{T}), \quad \text{where} \quad \mathbf{T} = \{tt, ff\}$$

  - $\mathcal{B}(\texttt{true})(s) = tt$;
  - $\mathcal{B}(\texttt{false})(s) = ff$;
  - $\mathcal{B}(a_1 = a_2)(s) = \begin{cases} tt, & \text{if } \mathcal{A}(a_1)(s) = \mathcal{A}(a_2)(s) \\ ff, & \text{otherwise;} \end{cases}$
  - similar for the other constructs.

"$=$" in the left hand side is a symbol, but "$=$" in the right hand side is the usual equality predicate on integers (and similar for "$\leq$", "$\neg$", and "$\wedge$").

# 1. Motivation

Denotational semantics:

- Interpretation function: interprets each statement as a partial function from $State$ into $State$

$$\mathcal{I}_{ds} : Stmt \rightarrow (State \rightsquigarrow State)$$

"$\rightsquigarrow$" means partial function. A partial function $f : A \rightsquigarrow B$ might not be defined for some input values $a \in A$. We will write $f(a) = \perp$ whenever $f$ is not defined for $a$ ("$\perp$" means undefined).

Denotational semantics:

- Meaning of "$x := a$"

$$\mathcal{I}_{ds}(x := a) : State \rightsquigarrow State$$

$$\mathcal{I}_{ds}(x := a)(s)(y) = \begin{cases} \mathcal{A}(a)(s), & \text{if } y = x \\ s(y), & \text{otherwise} \end{cases}$$

for all variables $y$.

- Meaning of "$\mathtt{skip}$"

$$\mathcal{I}_{ds}(skip) : State \rightsquigarrow State$$

$$\mathcal{I}_{ds}(skip) = id,$$

where $id$ is the identity function, i.e., the function which returns the same value that was used as its argument.

# 1. Motivation

Denotational semantics:

- Meaning of "$S_1; S_2$"

$$\mathcal{I}_{ds}(S_1; S_2) : State \rightsquigarrow State$$

$$\mathcal{I}_{ds}(S_1; S_2) = \mathcal{I}_{ds}(S_2) \circ \mathcal{I}_{ds}(S_1),$$

where "$\circ$" denotes the composition of (partial) functions.

Composition of two partial functions

$$A \overset{f}{\rightsquigarrow} B \overset{g}{\rightsquigarrow} C$$

is undefined for all input values $a \in A$ such that $f(a) = \perp$ or $g(f(a)) = \perp$.

Denotational semantics:

- Meaning of "$\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2$"

$$\mathcal{I}_{ds}(\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2) : State \rightsquigarrow State$$

$$\mathcal{I}_{ds}(\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2)(s) = \begin{cases} \mathcal{I}_{ds}(S_1)(s), & \text{if } \mathcal{B}(b)(s) = tt \\ \mathcal{I}_{ds}(S_2)(s), & \text{otherwise} \end{cases}$$

If $\mathcal{B}(b)(s) = tt$ for all states $s$ and $\mathcal{I}_{ds}(S_1)$ is a total function, then

$$\mathcal{I}_{ds}(\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2)$$

is a total function ($\mathcal{I}_{ds}(S_2)$ could be a partial function).

Denotational semantics:

- Meaning of "$\texttt{while}\, b\, \texttt{do}\, S$"

$$\mathcal{I}_{ds}(\texttt{while}\, b\, \texttt{do}\, S) : State \rightsquigarrow State$$

$$\mathcal{I}_{ds}(\texttt{while}\, b\, \texttt{do}\, S)(s) = \begin{cases} (\mathcal{I}_{ds}(\texttt{while}\, b\, \texttt{do}\, S) \circ \mathcal{I}_{ds}(S))(s), & \text{if } \mathcal{B}(b)(s) = tt \\ id(s), & \text{otherwise} \end{cases}$$

$\mathcal{I}_{ds}(\texttt{while}\, b\, \texttt{do}\, S)$ verifies an equation of the form $g = F(g)$. That is, $\mathcal{I}_{ds}(\texttt{while}\, b\, \texttt{do}\, S)$ should be a fixpoint of some function $F$. Given a function $F$, does it have fixpoints? If it has more than one, which one should we consider in order to get the meaning of $\texttt{while}\, b\, \texttt{do}\, S$?

Conclusions:

- All the constructs, except for `while do`, interpret to total functions;
- `while do` may interpret to partial functions;
- Defining the semantics of `while do` is a major task, and an adequate formalism is needed.

Denotational semantics is based on complete partially ordered sets, continuous functions, and fixed points.

# 2. Complete posets

A poset $M = (A; \leq)$ is called complete if $sup(L)$ exists for each chain $L$ of $M$.

**Remark 1** $M$ is complete iff:

1. $M$ has a least element $\perp_M$ (this is equivalent to existence of $sup(\emptyset)$);
2. $sup(L)$ exists for each non-empty chain $L$ of $M$

There is another completeness concept, namely by directed sets. It can be proved that completeness by directed sets and completeness by chains are equivalent.

# 2. Complete posets

Examples of complete posets:

1. Every poset which has a least element and contains only finite chains is complete.
2. Flat posets are complete.
3. $(\mathbf{N}; \leq)$ is not complete.
4. Given two sets $A$ and $B$, define the binary relation $\leq$ on $(A \rightsquigarrow B)$ by

$$f \leq g \quad \Leftrightarrow \quad \begin{array}{l} Dom(f) \subseteq Dom(g), \text{ and} \\ (\forall x \in Dom(f))(f(x) = g(x)), \end{array}$$

   for all $f, g \in (A \rightsquigarrow B)$.

   $((A \rightsquigarrow B); \leq)$ is a complete poset.

**Lemma 1** Let $M$ and $M'$ be two isomorphic poset. Then, $M$ is complete iff $M'$ is complete.

Let $A$ be a set and $(B;\leq)$ be a poset. Define the binary relation $\leq_{(A\to B)}$ on $(A\to B)$ by

$$f \leq_{(A\to B)} g \quad \Leftrightarrow \quad (\forall x \in A)(f(x) \leq g(x)),$$

for all $f, g \in (A\to B)$.

Let $S \subseteq (A\to B)$ and $a \in A$. $S(a)$ stands for

$$S(a) = \{f(a) | f \in S\}.$$

$S(a) = \emptyset$ if $S = \emptyset$.

**Lemma 2** Let $A$ be a set and $(B;\leq)$ a poset. Then, for any non-empty subset $S \subseteq (A \to B)$ the following property holds true:

$$\exists sup(S) \quad \Leftrightarrow \quad (\forall a \in A)(\exists sup(S(a))).$$

Moreover, if $sup(S)$ exists then

$$(\forall a \in A)(((sup(S))(a) = sup(S(a))).$$

**Theorem 1** Let $A$ be a set and $(B;\leq)$ be a poset. If $(B;\leq)$ is complete, then $((A{\to}B);\leq_{(A\to B)})$ is complete.

Let $L = \{f_i | i \geq 0\} \subseteq (\mathbf{N}_\perp \rightarrow \mathbf{N}_\perp)$ be the chain:

| $L$ | $\perp$ | 0 | 1 | 2 | 3 | $\cdots$ |
|---|---|---|---|---|---|---|
| $f_0$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\cdots$ |
| $f_1$ | $\perp$ | 1 | $\perp$ | $\perp$ | $\perp$ | $\cdots$ |
| $f_2$ | $\perp$ | 1 | 1! | $\perp$ | $\perp$ | $\cdots$ |
| $f_3$ | $\perp$ | 1 | 1! | 2! | $\perp$ | $\cdots$ |
| $\cdots$ | $\cdots$ | | | | | |

$sup(L)$ is the factorial function

$$sup(L)(x) = \begin{cases} 1, & x = 0 \\ x!, & x \in \mathbf{N} \\ \perp, & x = \perp, \end{cases}$$

Therefore, $L$ is an approximation of the factorial function:

$$f_0 \leq_{(\mathbf{N}_\perp \rightarrow \mathbf{N}_\perp)} f_1 \leq_{(\mathbf{N}_\perp \rightarrow \mathbf{N}_\perp)} f_2 \leq_{(\mathbf{N}_\perp \rightarrow \mathbf{N}_\perp)} \leq \ \cdots \ \leq sup(L)$$

Operations with complete posets: union

Union of disjoint complete posets is not a complete poset.

Reasons:

1. there is no least element.

Union of posets may not be a poset!

Operations with complete posets: intersection

Intersection of complete posets may not be a complete poset.

Reasons:

1. a least element may not exist, or
2. least upper bounds of some non-empty chains may not exist.

Intersection of posets is a poset!

Operations with complete posets: cartesian product

$$(A_1; \leq_1) \times (A_2; \leq_2) = (A_1 \times A_2; \leq)$$

where

$$(a_1, a_2) \leq (a'_1, a'_2) \quad \Leftrightarrow \quad a_1 \leq_1 a_2 \text{ and } a_2 \leq_2 a'_2$$

Cartesian product of complete posets is a complete poset.

A sub-poset $M' = (A'; \leq')$ of a poset $M = (A; \leq)$ is called a complete sub-poset of $M$ if it preserves $\perp_M$ and suprema of non-empty chains, i.e.,

1. $\perp_M \in A'$;
2. $(\forall L \subseteq A')(\text{L non-empty chain} \quad \Rightarrow \quad sup_M(L) \in A')$.

complete sub-poset $\neq$ sub-poset which is a complete poset in its own right. Reasons:

1. complete sub-poset: $\perp_{M'} = \perp_M$
   sub-poset which is complete: $\perp_{M'} \geq \perp_M$;
2. complete sub-poset: $sup_{M'}(L) = sup_M(L)$
   sub-poset which is complete: $sup_{M'}(L) \geq sup_M(L)$.

Let $M = (A, \leq)$ and $M' = (A', \leq')$ be posets. If $M'$ is complete, then supremum of any non-empty chain of monotone functions from $M$ to $M'$ is a monotone function from $M$ to $M'$.

$(A \rightarrow_m A')$ stands for the set of all monotone functions from $M$ to $M'$.

**Theorem 2** $(A \rightarrow_m A')$ is a complete sub-poset of $(A \rightarrow A')$ (w.r.t. $\leq_{(A \rightarrow A')}$).

# 3. Continuous Functions

In analysis, a function is continuous if it preserves limits. In the context in which a computation is modeled as the supremum of a chain, it is natural to consider a function as being continuous if it is compatible with the formation of suprema.

Let $M = (A; \leq)$ and $M' = (A'; \leq')$ be two complete posets and $f : A \rightarrow A'$ a function. $f$ is called continuous if

1. $sup(f(L))$ exists, and
2. $f(sup(L)) = sup(f(L))$,

for any non-empty chain $L$ of $M$.

**Theorem 3** (Continuity and monotony)
Let $M = (A; \leq)$ and $M' = (A'; \leq')$ be two complete posets and $f : A \rightarrow A'$ a function. Then, $f$ is continuous iff

1. $f$ is monotone;
2. $f(sup(L)) \leq' sup(f(L))$, for any non-empty chain $L$ in $M$.

If $M$ has only finite chains, then continuity is equivalent to monotony!

- $L : \quad a_1 \leq a_2 \leq \cdots \leq a_n$
- $f(L) : \quad f(a_1) \leq' f(a_2) \leq' \cdots \leq' f(a_n)$
- Then, $sup(L) = a_n$ and $sup(f(L)) = f(a_n) = f(sup(L))$.

In general, monotony does not imply continuity.

Let $\varphi : (\mathbf{N}_\perp \to \mathbf{N}_\perp) \to \mathbf{N}_\perp$ given by

$$\varphi(f) = \begin{cases} 1, & (\forall n \in \mathbf{N})(f(n) \neq \perp) \\ \perp, & \text{otherwise} \end{cases}$$

for all $f \in (\mathbf{N}_\perp \to \mathbf{N}_\perp)$.

It is easy to see that $\varphi$ is a monotone function.

Let $L$ be the chain

| $L$ | $\perp$ | 0 | 1 | 2 | 3 | $\cdots$ |
|---|---|---|---|---|---|---|
| $f_0$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\cdots$ |
| $f_1$ | $\perp$ | 0 | $\perp$ | $\perp$ | $\perp$ | $\cdots$ |
| $f_2$ | $\perp$ | 0 | 0 | $\perp$ | $\perp$ | $\cdots$ |
| $\cdots$ | $\cdots$ | | | | | |
| $sup(L)$ | $\perp$ | 0 | 0 | 0 | 0 | $\cdots$ |

$\varphi$ is not continuous because

$$sup(\varphi(L)) = sup(\{\perp\}) = \perp \neq 1 = \varphi(sup(L)).$$

Examples of continuous functions:

1. Let $M_i = (A_i; \leq_i)$ be flat posets ($1 \leq i \leq n$) and let $M = (A; \leq)$ be a complete poset. Any monotone function $f : A_1 \times \cdots \times A_n \rightarrow A$ is continuous.
2. Constant functions (defined on complete posets) are continuous.
3. Identity functions (defined on complete posets) are continuous.
4. Projection functions (defined on complete posets) are continuous:

$$pr_i : A_1 \times \cdots \times A_n \rightarrow A_i$$

$pr_i(a_1, \ldots, a_i, \ldots, a_n) = a_i.$

**Theorem 4** Composition of continuous functions is a continuous function.

Consequences ($M = (A; \leq)$ and $M_i = (A_i; \leq_i)$ are complete posets):

1. $f : A \rightarrow A_1 \times \cdots \times A_n$ is continuous iff $pr_i \circ f$ is continuous, for all $1 \leq i \leq n$.

   In other words, $f$ is continuous iff it is continuous in each coordinate.

2. Let $f_i : A \rightarrow A_i$, $1 \leq i \leq n$, and $f : A \rightarrow A_1 \times \cdots \times A_n$ given by $f(a) = (f_1(a), \ldots, f_n(a))$, for all $a \in A$.

   $f$ is continuous iff $f_i$ is continuous, for all $i$.

Let $f : A_1 \times \cdots \times A_n \to A$ be a function, where $n \geq 2$. The Curry function associated to $f$ is the function

$$f^c : A_1 \times \cdots \times A_{n-1} \to (A_n \to A)$$

given by

$$f^c(a_1, \ldots, a_{n-1})(a_n) = f(a_1, \ldots, a_n),$$

for all $(a_1, \ldots, a_{n-1}) \in A_1 \times \cdots \times A_{n-1}$ and $a_n \in A_n$.

$f^c(a_1, \ldots, a_{n-1}) : A_n \to A$ will also be called Curry function associated to $f$, for any $(a_1, \ldots, a_{n-1}) \in A_1 \times \cdots \times A_{n-1}$.

**Theorem 5** (Monotony and Curry functions)
Let $M = (A; \leq), M_1 = (A_1; \leq_1), \ldots, M_n = (A_n; \leq_n)$ be posets and $f : A_1 \times \cdots \times A_n \to A$ be a function, where $n \geq 2$. Then, $f$ is monotone iff all its associated Curry functions are monotone.

**Theorem 6** (Continuity and Curry functions)
Let $M = (A; \leq), M_1 = (A_1; \leq_1), \ldots, M_n = (A_n; \leq_n)$ be complete posets and $f : A_1 \times \cdots \times A_n \to A$ be a function, where $n \geq 2$. Then, $f$ is continuous iff all its associated Curry functions are continuous.

Let $M_1 = (A_1; \leq_1)$ and $M_2 = (A_2; \leq_2)$ be complete posets. Denote by $[M_1 \rightarrow M_2]$ (or $[A_1 \rightarrow A_2]$) the set of all continuous functions from $M_1$ to $M_2$.

Let $\psi : [A_1 \rightarrow A_2] \times A_1 \rightarrow A_2$ given by

$$\psi(f, a) = f(a),$$

for all $f \in [A_1 \rightarrow A_2]$ and $a \in A_1$. By using Curry functions we can prove that $f$ is continuous.

$\psi$ is called the apply function.

Let $M = (A, \leq)$ and $M' = (A', \leq')$ be complete posets. Then, supremum of any non-empty chain of continuous functions from $M$ to $M'$ is a continuous function from $M$ to $M'$.

**Theorem 7** $[A \rightarrow A']$ is a complete sub-poset of $(A \rightarrow_m A')$.

$\subseteq_{csp}$ stands for complete sub-poset

$$[A \rightarrow A'] \subseteq_{csp} (A \rightarrow_m A') \subseteq_{csp} (A \rightarrow A')$$

# 4. Fixed Points

Let $f : A \to A$. If $f(a) = a$ then $a$ is called a fixed point of $f$.

Let $M = (A; \leq)$ be a complete poset and $f : A \to A$ a continuous function. Then

$$L : \quad \perp, f(\perp), f(f(\perp)), \ldots$$

is a chain.

There exists $sup(L)$ because $M$ is complete and

$$f(sup(L)) = sup(f(L)) = sup(\{f(\perp), f(f(\perp)), \ldots\}) = sup(L)$$

Therefore, $sup(L)$ is a fixed point of $f$

# 4. Fixed Points

$sup(L)$ is the least fixed point of $f$. Indeed,

- let $a$ be a fixed point of $f$ (i.e., $f(a) = a$)
- $\perp \leq a$
- $f(\perp) \leq f(a) = a$, because $f$ is monotone
- $f(f(\perp)) \leq f(a) = a$ etc.
- $a$ is an upper bound of $L$ and, therefore, $sup(L) \leq a$ showing that $sup(L)$ is the least fixed point of $f$.

**Theorem 8** (Fixed Point Theorem; Knaster-Tarski)
Let $M = (A; \leq)$ be a complete poset and $f : A \rightarrow A$ be a continuous function. Then $f$ has a least fixed point, denoted $\mu(f)$. Moreover,

$$\mu(f) = sup(\{f^i(\perp) | i \geq 0\}),$$

where $f^0(x) = x$ and $f^{i+1}(x) = f(f^i(x))$, for all $i \geq 0$ and $x \in A$.

The fixed point theorem provides us with a method for computing the least fixed point of a continuous function defined on a complete poset.

Let $M = (A; \leq)$ be a complete poset and $P : A \rightarrow \{0, 1\}$ be a predicate. $P$ is called admissible if

$$(\forall L \subseteq A)(L \text{ non-empty chain} \wedge (\forall a \in L)(P(a)) \Rightarrow P(sup(L)))$$

**Theorem 9** (Fixed Point Induction; D. Scott)
Let $M = (A; \leq)$ be a complete poset, $f : A \rightarrow A$ a continuous function, and $P$ a predicate on $A$. If:

- $P$ is admissible;
- $P(\perp)$
- $P(f^i(\perp)) \Rightarrow P(f^{i+1}(\perp))$, for all $i \geq 0$

then $P(\mu(f))$.

Admissible predicates:

- If $P$ and $Q$ are admissible, then $P \vee Q$ and $P \wedge Q$ are admissible.
- If $f_i, g_i : A \to A'$ are continuous, $1 \leq i \leq n$, then
$$P(a) \quad \Leftrightarrow \quad (\forall i)(f_i(a) \leq' g_i(a))$$
is admissible ($P : A \to \{0, 1\}$).
- If $f, g : A \to A'$ are continuous, then
$$P(a) \quad \Leftrightarrow \quad f(a) = g(a)$$
is admissible ($P : A \to \{0, 1\}$).

A general approach to `while` programs:

Let $\mathcal{B} = (\mathcal{V}, \mathcal{F}, \mathcal{P})$ be a basis ($\mathcal{V}$ is a set of variables, $\mathcal{F}$ is a set of function symbols, and $\mathcal{P}$ is a set of predicate symbols).

- if $x \in \mathcal{V}$ and $t$ is a term over $\mathcal{B}$, then $x := t$ is a while program over $\mathcal{B}$;
- if $S_1$ and $S_2$ are while programs over $\mathcal{B}$ and $e$ is a logical expression over $\mathcal{B}$, then $S_1; S_2$, $if\ e\ then\ S_1\ else\ S_2$ and $while\ e\ do\ S_1$ are while programs over $\mathcal{B}$.

Convention: when $S$ is of the form "$S_1; S_2$", we will write $if\ e\ then\ S_1\ else\ (S)$ instead of $if\ e\ then\ S_1\ else\ S$, and $while\ e\ do\ (S)$ instead of $while\ e\ do\ S$.

Examples:

1. $while\ x > 0\ do\ x := x - 1$.

2. $y := 1;\ while\ \neg(x = 1)\ do\ (y := y * x;\ x := x - 1)$.

3. $z := 0;\ while\ y \leq x\ do\ (z := z + 1;\ x := x - y)$.

# 5. Denotational Semantics of While Programs

Let $\mathcal{B} = (\mathcal{V}, \mathcal{F}, \mathcal{P})$ be a basis. An interpretation of $\mathcal{B}$ is any pair $\mathcal{I} = (D, \mathcal{I}_0)$ consisting of a non-empty domain $D$ and an initial interpretation $\mathcal{I}_0$ satisfying:

- $\mathcal{I}_0(f) : D^n \rightarrow D$, for any function symbol $f$ of arity $n \geq 0$;
- $\mathcal{I}_0(P) : D^n \rightarrow Bool$, for any predicate symbol $P$ of arity $n \geq 0$, where $Bool = \{0, 1\}$.

Assignment: $\gamma : \mathcal{V} \rightarrow D$. Let $\Gamma$ be the set of all assignments.

Let $S$ be a while program over a basis $\mathcal{B}$ and $\mathcal{I}$ an interpretation of $\mathcal{B}$. The semantic function associated to $S$ under the interpretation $\mathcal{I}$ is the function

$$\phi_{\mathcal{I}}(S) : \Gamma_{\perp} \rightarrow \Gamma_{\perp}$$

given by:

- $\phi_{\mathcal{I}}(S)(\gamma) = \begin{cases} \gamma[x/\mathcal{I}(t)(\gamma)], & \text{if } \gamma \neq \perp \\ \perp, & \text{if } \gamma = \perp, \end{cases}$

  for any $\gamma \in \Gamma_{\perp}$, if $S$ is the program $x := t$;

- $\phi_{\mathcal{I}}(S) = \phi_{\mathcal{I}}(S_2) \circ \phi_{\mathcal{I}}(S_1)$, if $S$ is the program $S_1; S_2$;

- $$\phi_{\mathcal{I}}(S)(\gamma) = \begin{cases} \phi_{\mathcal{I}}(S_1)(\gamma), & \text{if } \mathcal{I}(e)(\gamma) = 1 \text{ and } \gamma \neq \perp \\ \phi_{\mathcal{I}}(S_2)(\gamma), & \text{if } \mathcal{I}(e)(\gamma) = 0 \text{ and } \gamma \neq \perp \\ \perp, & \text{if } \gamma = \perp, \end{cases}$$

  for any $\gamma \in \Gamma_\perp$, if $S$ is the program $if\ e\ then\ S_1\ else\ S_2$;

- $\phi_{\mathcal{I}}(S) = \mu(F)$, if $S$ is the program $while\ e\ do\ S_1$, where $F$ is the function

  $$F : [\Gamma_\perp \to \Gamma_\perp] \to [\Gamma_\perp \to \Gamma_\perp]$$

  given by

  $$F(f)(\gamma) = \begin{cases} f(\phi_{\mathcal{I}}(S_1)(\gamma)), & \text{if } \mathcal{I}(e)(\gamma) = 1 \text{ and } \gamma \neq \perp \\ \gamma, & \text{if } \mathcal{I}(e)(\gamma) = 0 \text{ and } \gamma \neq \perp \\ \perp, & \text{if } \gamma = \perp, \end{cases}$$

  for any $f \in [\Gamma_\perp \to \Gamma_\perp]$ and $\gamma \in \Gamma_\perp$.

Constant propagation is an analysis that determines whether an expression always evaluates to a constant value.

- $x := 5;\ y := x * x + 25$
- $y$ will always be 50
- it is safe to replace the above statements by

$$x := 5;\ y := 50$$

Detection of sign analysis determines the sign of expressions

- $x := 5;\ y := x * x + 25$
- $y$ will always be positive (independently of the value assigned to $x$)
- this property is useful for code elimination. In a statement as

$$y := x * x + 25;\ \textit{while}\ y \leq 0\ \textit{do}\ \ldots$$

there is no need to generate code for the while-loop because it is never executed

Dependency analysis regards some of the variables as input variables and others as output variables, and determines whether or not the final values of the output variables depend upon the initial values of the input variables.

- $x$ is an input variable and $y$ is an output variable
- $x := 5;\ y := x * x + 25$ – in this program there is a functional dependency between the input and the output variables;
- $y := 1;\ while\ \neg(x = 1)\ do\ (y := y * x;\ x := x - 1)$ – the final value of $y$ depends upon the initial value of $x$;
- $while\ \neg(x = 1)\ do\ (y := y * x;\ x := x - 1)$ – the final value of $y$ does not only depend upon the initial value of $x$, but also on the initial value of $y$.

# 6. Static Analysis

In order to develop dependency analysis we will use two symbols, $OK$ and $D?$, with the following meaning:

- when $OK$ labels a given value (or variable) then the value definitely depends on the initial values of the input variables, and only on them;

- when $D?$ labels a given value then the value may depend on the initial values of non-input variables ($D =$ dubious)

- $\mathbf{P} = \{OK, D?\}$, and structure this set as a complete poset by $OK \leq D?$.

At any state, all variables will be labeled by an element in $\mathbf{P}$. We will consider one more variable, `on-track`, which gives information about the "flow of control". It will be labeled by elements in $\mathbf{P}$ too.

- A p-state is any function $\psi : \mathcal{V} \cup \{\mathtt{on-track}\} \rightarrow \mathbf{P}$
- $\Psi$ denotes the set of all p-states. It is a complete poset
- A p-state is called proper if $\psi(\mathtt{on-track}) = OK$; otherwise, it is caled improper
- $OK(\psi) = \{x | \psi(x) = OK\}$
- $LOST$ is the p-state given by $LOST(x) = D?$, for all $x$.

The p-interpretation function of terms and logical expression is defined as follows:

- $$p\mathcal{I}(t)(\psi) = \begin{cases} OK, & \text{if } \psi \text{ is proper} \\ D?, & \text{otherwise,} \end{cases}$$

  if $t \in \mathcal{F}$ is a constant symbol;

- $$p\mathcal{I}(t)(\psi) = \begin{cases} \psi(t), & \text{if } \psi \text{ is proper} \\ D?, & \text{otherwise,} \end{cases}$$

  if $t \in \mathcal{V} \cup \{\texttt{on} - \texttt{track}\}$ is a variable;

- $p\mathcal{I}(f(t_1, \ldots, t_n))(\psi) = sup(\{p\mathcal{I}(t_1)(\psi), \ldots, p\mathcal{I}(t_n)(\psi)\});$

- $p\mathcal{I}(true)(\psi)$, $p\mathcal{I}(false)(\psi)$ and $p\mathcal{I}(p)(\psi)$, for any propositional constant $p$, are defined as for constant symbols;

- $p\mathcal{I}(t_1 = t_2)(\psi) = sup(\{p\mathcal{I}(t_1)(\psi), p\mathcal{I}(t_2)(\psi)\})$;

- $p\mathcal{I}(P(t_1, \ldots, t_n))(\psi) = sup(\{p\mathcal{I}(t_1)(\psi), \ldots, p\mathcal{I}(t_2)(\psi)\})$;

- $p\mathcal{I}(\neg e)(\psi) = p\mathcal{I}(e)(\psi)$;

- $p\mathcal{I}(e_1\, o\, e_2)(\psi) = sup(\{p\mathcal{I}(e_1)(\psi), p\mathcal{I}(e_2)(\psi)\})$, for any $o \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$,

for any $\psi \in \Psi$.

The p-interpretation function of programs is defined as follows:

- $p\phi_{\mathcal{I}}(S)(\psi) = \psi[x/p\mathcal{I}(t)(\psi)]$, for any $\psi \in \Psi$, if $S$ is $x := t$;

- $p\phi_{\mathcal{I}}(S) = p\phi_{\mathcal{I}}(S_2) \circ p\phi_{\mathcal{I}}(S_1)$, if $S$ is $S_1; S_2$;

- $\phi_{\mathcal{I}}(S)(\psi) = \begin{cases} sup(\{p\phi_{\mathcal{I}}(S_1)(\psi), p\phi_{\mathcal{I}}(S_2)(\psi)\}), & \text{if } p\mathcal{I}(e)(\psi) = OK \\ LOST, & \text{otherwise,} \end{cases}$

  for any $\psi \in \Psi$, if $S$ is $if\ e\ then\ S_1\ else\ S_2$;

- $p\phi_{\mathcal{I}}(S) = \mu(F)$, if $S$ is $while\ e\ do\ S_1$, where $F$ is the function

$$F : [\Psi \to \Psi] \to [\Psi \to \Psi]$$

given by

$$F(f)(\psi) = \begin{cases} sup(\{f(p\phi_{\mathcal{I}}(S_1)(\psi)), \psi\}) & \text{if } p\mathcal{I}(e)(\psi) = OK \\ LOST, & \text{otherwise,} \end{cases}$$

for any $f \in [\Psi \to \Psi]$ and $\psi \in \Psi$.

**Input:**

while program $S$;

sets $I$ and $O$ of input and output variables

**Output:**

$Yes$ – if there definitely is a functional dependency

$No?$ – if there may not be a functional dependency

**Begin**

let $\psi_0$ given by $OK(\psi_0) = I \cup \{\mathtt{on-track}\}$;

let $\psi_f := p\phi_{\mathcal{I}}(S)(\psi_0)$;

if $O \cup \{\mathtt{on-track}\} \subseteq OK(\psi_f)$ then $Yes$ else $No?$

**End.**

## Free Variables

- $FV(x := t) = \{x\} \cup FV(t)$;
- $FV(S_1; S_2) = FV(S_1) \cup FV(S_2)$;
- $FV(if\ b\ then\ S_1\ else\ S_2) = FV(b) \cup FV(S_1) \cup FV(S_2)$;
- $FV(while\ b\ do\ S) = FV(b) \cup FV(S)$.

**Theorem 10**

(1) For any while program *while b do S* the following holds true:

$$p\phi_X(\textit{while } b \textit{ do } S) = F^{m+1}(f_0),$$

where $X = FV(\textit{while } b \textit{ do } S)$, $f_0$ is the least element of $[\Psi_X \to \Psi_X]$ and $m = |X|$.

(2) There exists a while program *while b do S* such that

$$p\phi_X(\textit{while } b \textit{ do } S) \neq F^{m-1}(f_0),$$

where $X = FV(\textit{while } b \textit{ do } S)$, $f_0$ is the least element of $[\Psi_X \to \Psi_X]$ and $m = |X|$.