Mathematics for Informatics

Carlos Areces and Patrick Blackburn

areces@loria.fr blackbur@loria.fr

http://www.loria.fr/~areces http://www.loria.fr/~blacbur

INRIA Lorraine
Nancy, France

2007/2008

The Course

The Programing Language $\mathscr{S}$

- imperative style, very simple

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \dots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \dots$
    - you don't need to declare them
    - they are all initialized to the value 0

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \ldots$
    - you don't need to declare them
    - they are all initialized to the value 0
- The only data type: the natural numbers (including 0)
  - there are no constants

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \ldots$
    - you don't need to declare them
    - they are all initialized to the value 0
- The only data type: the natural numbers (including 0)
  - there are no constants
- increment in one: $X_1 \leftarrow X_1 + 1$

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \ldots$
    - you don't need to declare them
    - they are all initialized to the value 0
- The only data type: the natural numbers (including 0)
  - there are no constants
- increment in one: $X_1 \leftarrow X_1 + 1$
- decrement in one:
  - $X_1 \leftarrow X_1 - 1$
  - if X_1 is 0 then it won't be modified

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \ldots$
    - you don't need to declare them
    - they are all initialized to the value 0
- The only data type: the natural numbers (including 0)
  - there are no constants
- increment in one: $X_1 \leftarrow X_1 + 1$
- decrement in one:
  - $X_1 \leftarrow X_1 - 1$
  - if X_1 is 0 then it won't be modified
- a primitive IF GOTO construction
  - IF $X_1 \neq 0$ GOTO A
  - A is a label that marks another instruction in the program

The Programing Language $\mathscr{S}$

- imperative style, very simple
  - input variables: $X_1, X_2, \ldots$
  - output variable (only one): Y
  - temporal variables: $Z_1, Z_2, \ldots$
    - you don't need to declare them
    - they are all initialized to the value 0
- The only data type: the natural numbers (including 0)
  - there are no constants
- increment in one: $X_1 \leftarrow X_1 + 1$
- decrement in one:
  - $X_1 \leftarrow X_1 - 1$
  - if X_1 is 0 then it won't be modified
- a primitive IF GOTO construction
  - IF $X_1 \neq 0$ GOTO A
  - A is a label that marks another instruction in the program
- you can't call subroutines.

Example 1

$$
\begin{array}{ll}
[A] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{IF } X \neq 0 \text{ GOTO } A
\end{array}
$$

- We write X for X_1; Z for Z_1
- when $X = 0$ the program stops as there is no next instruction
- it comput the function $f : \mathbb{N} \to \mathbb{N}$,

$$f(x) = \begin{cases} x & \text{if } x \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

- it always left the variable X in 0

Example 2

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } E \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } A
\end{array}
$$

- it computes the function $f : \mathbb{N} \to \mathbb{N}, f(x) = x$

Example 2

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } E \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } A
\end{array}
$$

- it computes the function $f : \mathbb{N} \to \mathbb{N}, f(x) = x$
- when it tries to go to E, it finishes

Example 2

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } E \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } A
\end{array}
$$

- it computes the function $f : \mathbb{N} \rightarrow \mathbb{N}, f(x) = x$
- when it tries to go to E, it finishes
- In the example, Z is used only to force an unconditional jump. In general GOTO L is equivalent to

$$
\begin{array}{l}
V \leftarrow V + 1 \\
\text{IF } V \neq 0 \text{ GOTO } L
\end{array}
$$

where V is a new variable.

Macros

- $\mathscr{S}$ does not have unconditional jump
- but we can simulate it with GOTO L

Macros

- $\mathscr{S}$ does not have unconditional jump
- but we can simulate it with GOTO L
- we use it as if it were part of the language, but
  - each time that we see
    $$\text{GOTO } L$$
    in a program P, we replace it with
    $$V \leftarrow V + 1$$
    $$\text{IF } V \neq 0 \text{ GOTO } L$$
    where V is a variable that does not appears in P.

Macros

- $\mathscr{S}$ does not have unconditional jump
- but we can simulate it with GOTO L
- we use it as if it were part of the language, but
  - each time that we see
    $$\text{GOTO } L$$
    in a program P, we replace it with
    $$V \leftarrow V + 1$$
    $$\text{IF } V \neq 0 \text{ GOTO } L$$
    where V is a variable that does not appears in P.

We will see that we can simulate many other operations.
Once we know that we can write them down in $\mathscr{S}$, we will used them as if they were part of the language (they are called pseudoinstructions)

- the abbreviated form it's called a macro
- the program that the macro stands for it's called macro expansion

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- the first cycle copies the value from X into Y and Z

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- the first cycle copies the value from X into Y and Z
- the second cycle puts in X the original value and leaves Z in zero.

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- the first cycle copies the value from X into Y and Z
- the second cycle puts in X the original value and leaves Z in zero.
- we use the macro GOTO A
  - it should not be expanded as

$$
\begin{array}{l}
Z \leftarrow Z + 1 \\
\text{IF } Z \neq 0 \text{ GOTO } A
\end{array}
$$

but as

$$
\begin{array}{l}
Z_2 \leftarrow Z_2 + 1 \\
\text{IF } Z_2 \neq 0 \text{ GOTO } A
\end{array}
$$

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- it can be used to assign to variable V the content of variable V' and leave V' without changes within an arbitrary program P: $V \leftarrow V'$.
  - change Y by V
  - change X by V'
  - change Z for a temporal variable that does not appears in P

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- it can be used to assign to variable V the content of variable V' and leave V' without changes within an arbitrary program P: $V \leftarrow V'$.
  - change Y by V
  - change X by V'
  - change Z for a temporal variable that does not appears in P
- but it works properly only when $V = 0$ and $Z = 0$

Macro for variable assignment: $V \leftarrow V'$

$$
\begin{array}{ll}
 & Y \leftarrow 0 \\
[A] & \text{IF } X \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & X \leftarrow X - 1 \\
 & Y \leftarrow Y + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{GOTO } C
\end{array}
$$

- it can be used to assign to variable V the content of variable V' and leave V' without changes within an arbitrary program P: $V \leftarrow V'$.
  - change Y by V
  - change X by V'
  - change Z for a temporal variable that does not appears in P
- but it works properly only when $V = 0$ and $Z = 0$
- we fix this by using $Y \leftarrow 0$ as first pseudoinstruction
  - we don't need to make $Z \leftarrow 0$

Macro for the assignment of zero: $V \leftarrow 0$

In a program P, the pseudoinstruction $V \leftarrow 0$ is expanded as

$$\begin{array}{ll} [L] & V \leftarrow V - 1 \\ & \text{IF } V \neq 0 \text{ GOTO } L \end{array}$$

where L is a label that does not appear in P

Macro for the assignment of zero: $V \leftarrow 0$

In a program P, the pseudoinstruction $V \leftarrow 0$ is expanded as

$$
\begin{array}{ll}
[L] & V \leftarrow V - 1 \\
& \text{IF } V \neq 0 \text{ GOTO } L
\end{array}
$$

where L is a label that does not appear in P

Then the program for $V \leftarrow V'$ is:

$$
\begin{array}{ll}
& \color{red}{V \leftarrow 0} \\
[A] & \text{IF } V \neq 0 \text{ GOTO } B \\
& \text{GOTO } C \\
[B] & V \leftarrow V - 1 \\
& V' \leftarrow V' + 1 \\
& Z \leftarrow Z + 1 \\
& \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
& \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
& V \leftarrow V + 1 \\
& \text{GOTO } C
\end{array}
$$

Macro for the assignment of zero: $V \leftarrow 0$

In a program P, the pseudoinstruction $V \leftarrow 0$ is expanded as

$$
\begin{array}{ll}
[L] & V \leftarrow V - 1 \\
 & \text{IF } V \neq 0 \text{ GOTO } L
\end{array}
$$

where L is a label that does not appear in P

Then the program for $V \leftarrow V'$ is:

$$
\begin{array}{ll}
[H] & V \leftarrow V - 1 \\
 & \text{IF } V \neq 0 \text{ GOTO } H \\
[A] & \text{IF } V \neq 0 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & V \leftarrow V - 1 \\
 & V' \leftarrow V' + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } D \\
 & \text{GOTO } E \\
[D] & Z \leftarrow Z - 1 \\
 & V \leftarrow V + 1 \\
 & \text{GOTO } C
\end{array}
$$

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[B] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z \leftarrow Z - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{GOTO } B
\end{array}
$$

Addition of two variables

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[B] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & Z \leftarrow Z - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{GOTO } B
\end{array}
$$

computes the function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$f(x_1, x_2) = x_1 + x_2$$

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & \text{IF } Y \neq 0 \text{ GOTO } B \\
 & \text{GOTO } A \\
[B] & Y \leftarrow Y - 1 \\
 & Z \leftarrow Z - 1 \\
 & \text{GOTO } C
\end{array}
$$

Substraction of two variables

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & \text{IF } Y \neq 0 \text{ GOTO } B \\
 & \text{GOTO } A \\
[B] & Y \leftarrow Y - 1 \\
 & Z \leftarrow Z - 1 \\
 & \text{GOTO } C
\end{array}
$$

computes the function
$f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$
g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ \uparrow & \text{otherwise} \end{cases}
$$

Substraction of two variables

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[C] & \text{IF } Z \neq 0 \text{ GOTO } A \\
 & \text{GOTO } E \\
[A] & \text{IF } Y \neq 0 \text{ GOTO } B \\
 & \text{GOTO } A \\
[B] & Y \leftarrow Y - 1 \\
 & Z \leftarrow Z - 1 \\
 & \text{GOTO } C
\end{array}
$$

computes the function
$f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$
g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{if } x_1 \geq x_2 \\ \uparrow & \text{otherwise} \end{cases}
$$

- g is a partial function
- we mark indefinition as $\uparrow$ (in the metalanguage)
- the cause of indefinition is no termination
  - there is no other cause of indefinition

States

A state in a program P is a list of equations of the form $V = m$ (where V is a variable and m is a number) such that

- there is exactly one equation for each variable that appears in P

States

A state in a program P is a list of equations of the form $V = m$ (where V is a variable and m is a number) such that

- there is exactly one equation for each variable that appears in P

For example, for P:

$$\begin{array}{ll} [A] & X \leftarrow X - 1 \\ & Y \leftarrow Y + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \end{array}$$

- the following are possible states of P:
  - $X = 3, Y = 1$
  - $X = 3, Y = 1, Z = 0$
  - $X = 3, Y = 1, Z = 8$
    - it is not necessary that the state is reachable

States

A state in a program P is a list of equations of the form $V = m$ (where V is a variable and m is a number) such that

- there is exactly one equation for each variable that appears in P

For example, for P:

$$\begin{array}{ll} [A] & X \leftarrow X - 1 \\ & Y \leftarrow Y + 1 \\ & \text{IF } X \neq 0 \text{ GOTO } A \end{array}$$

- the following are possible states of P:
  - $X = 3, Y = 1$
  - $X = 3, Y = 1, Z = 0$
  - $X = 3, Y = 1, Z = 8$
    - it is not necessary that the state is reachable
- the following are not states of P:
  - $X = 3$
  - $X = 3, Z = 0$
  - $X = 3, Y = 1, X = 0$

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \ldots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \ldots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \dots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \dots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \dots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$
3. if the i-th instruction of P is IF $V \neq 0$ GOTO L
   - τ is identical to σ

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \dots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$
3. if the i-th instruction of P is IF $V \neq 0$ GOTO L
   - τ is identical to σ

   3.1 if σ has $V = 0$ then $j = i + 1$

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \dots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$
3. if the i-th instruction of P is IF $V \neq 0$ GOTO L
   - τ is identical to σ

   3.1 if σ has $V = 0$ then $j = i + 1$
   3.2 if σ has $V = m$ for $m \neq 0$ then

Instant Description

Let's assume that the program P has length n.

For a state σ of P and $i \in \{1, \ldots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$
3. if the i-th instruction of P is IF $V \neq 0$ GOTO L
   - τ is identical to σ

   3.1 if σ has $V = 0$ then $j = i + 1$

   3.2 if σ has $V = m$ for $m \neq 0$ then
   - if there is an instruction in P with label L then $j = \min\{k : k\text{-th instruction in } P \text{ with label } L\}$

Instant Description

Let's assume that the program P has length n.
For a state σ of P and $i \in \{1, \ldots, n+1\}$,

- the pair (i, σ) is an instant description of P.
- (i, σ) is called terminal if $i = n + 1$.

For (i, σ) non terminal, we can define the succesor state (j, τ) as:

1. if the i-th instruction of P is $V \leftarrow V + 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = m + 1$
2. if the i-th instruction of P is $V \leftarrow V - 1$.
   - $j = i + 1$
   - τ is σ, except that $V = m$ is replaced by $V = \max\{m - 1, 0\}$
3. if the i-th instruction of P is IF $V \neq 0$ GOTO L
   - τ is identical to σ

   3.1 if σ has $V = 0$ then $j = i + 1$
   3.2 if σ has $V = m$ for $m \neq 0$ then
      - if there is an instruction in P with label L then $j = \min\{k : k\text{-th instruction in } P \text{ with label } L\}$
      - otherwise $j = n + 1$

Computations

A computation of a program P from an instant description d_1 is a list

$$d_1, d_2, \ldots, d_k$$

of instant descriptions of P such that

- d_{i+1} is the successor of d_i for $i \in \{1, 2, \ldots, k-1\}$
- d_k is terminal

States and Instant Descriptions

Given a program P and let $r_1, \ldots, r_m$ be numbers.

- The initial state of P for $r_1, \ldots, r_m$ is the state σ_1, that has

  $$X_1 = r_1 \quad , \quad X_2 = r_2 \quad , \quad \ldots \quad , \quad X_m = r_m \quad , \quad Y = 0$$

  together with

  $$V = 0$$

  for each variable V that appears in P which is different from $X_1, \ldots, X_m, Y$
- the initial description of P for $r_1, \ldots, r_m$ is

  $$(1, \sigma_1)$$

Computation from the initial state

Let P be a program and let

- $r_1, \ldots, r_m$ be numbers
- σ_1 the initial state (the P and $r_1, \ldots, r_m$)

Computation from the initial state

Let P be a program and let

- $r_1, \ldots, r_m$ be numbers
- σ_1 the initial state (the P and $r_1, \ldots, r_m$)

There are two cases

- there is a computation of P

$$d_1, \ldots, d_k$$

  such that $d_1 = (1, \sigma_1)$
  We note as $\Psi_P^{(m)}(r_1, \ldots, r_m)$ the value of Y in the instant configuration d_k.

Computation from the initial state

Let P be a program and let

- $r_1, \ldots, r_m$ be numbers
- σ_1 the initial state (the P and $r_1, \ldots, r_m$)

There are two cases

- there is a computation of P

  $$d_1, \ldots, d_k$$

  such that $d_1 = (1, \sigma_1)$

  We note as $\Psi_P^{(m)}(r_1, \ldots, r_m)$ the value of Y in the instant configuration d_k.
- there is no computation, i.e. there is an infinite sequence

  $$d_1, d_2, d_3, \ldots$$

  where
  - $d_1 = (1, \sigma_1)$.
  - d_{i+1} is the successor of d_1

  We say that $\Psi_P^{(m)}(r_1, \ldots, r_m)$ is undefined (we note $\Psi_P^{(m)}(r_1, \ldots, r_m) \uparrow$)

Computable Functions

A (partial) function $f : \mathbb{N}^m \rightarrow \mathbb{N}$ is partially computable if there is a program P such that

$$f(r_1, \ldots, r_m) = \Psi_P^{(m)}(r_1, \ldots, r_m)$$

for each $(r_1, \ldots, r_m) \in \mathbb{N}^m$.

Computable Functions

A (partial) function $f : \mathbb{N}^m \to \mathbb{N}$ is partially computable if there is a program P such that

$$f(r_1, \ldots, r_m) = \Psi_P^{(m)}(r_1, \ldots, r_m)$$

for each $(r_1, \ldots, r_m) \in \mathbb{N}^m$.

The equality (in the meta-language) is true iff

- both sides are defined and they have the same value, or
- both sides are undefined

Computable Functions

A (partial) function $f : \mathbb{N}^m \to \mathbb{N}$ is partially computable if there is a program P such that

$$f(r_1, \ldots, r_m) = \Psi_P^{(m)}(r_1, \ldots, r_m)$$

for each $(r_1, \ldots, r_m) \in \mathbb{N}^m$.

The equality (in the meta-language) is true iff

- both sides are defined and they have the same value, or
- both sides are undefined

A function f is computable if it is partically computable and total.

Computable Functions

A (partial) function $f : \mathbb{N}^m \to \mathbb{N}$ is partially computable if there is a program P such that

$$f(r_1, \ldots, r_m) = \Psi_P^{(m)}(r_1, \ldots, r_m)$$

for each $(r_1, \ldots, r_m) \in \mathbb{N}^m$.

The equality (in the meta-language) is true iff

- both sides are defined and they have the same value, or
- both sides are undefined

A function f is computable if it is partically computable and total.

Note that the same program P can be used to compute functions with 1 variable, 2 variables, etc. Suppose that in P we have occurrences of X_n but not of X_i for $i > n$.

- if we only specify $m < n$ input variables , $X_{m+1}, \ldots, X_n$ take the value 0
- if we specify $m > n$ input variables, P will ignore $X_{n+1}, \ldots, X_m$