Model Checking I
alias
Reactive Systems Verification

Luca Tesei

MSc in Computer Science, University of Camerino

Topics

- Program Graphs
- Semantics of Program Graphs as Transition Systems

Material

Reading:

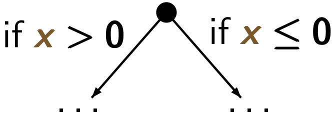
Chapter 2 of the book, pages 29–35.

More:

The slides in the following pages are taken from the material of the course “Introduction to Model Checking” held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

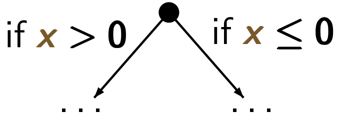
Data-dependent systems

problem: TS-representation of conditional branchings **?**



Data-dependent systems

problem: TS-representation of conditional branchings **?**

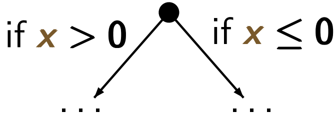


example: sequential program

```
WHILE $x > 0$ DO
    $x := x - 1$;
    $y := y + 1$
OD
...
```

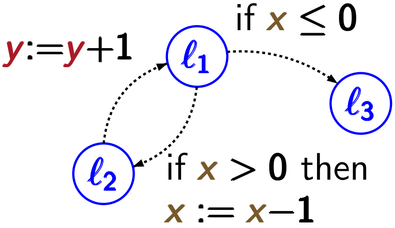
Data-dependent systems

problem: TS-representation of conditional branchings **?**



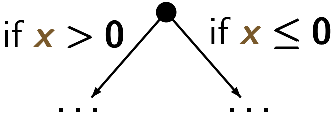
example: sequential program

```
WHILE  $x > 0$ DO
       $x := x-1$;
       $y := y+1$
OD
...
```



Data-dependent systems

problem: TS-representation of conditional branchings **?**



if $x > 0$

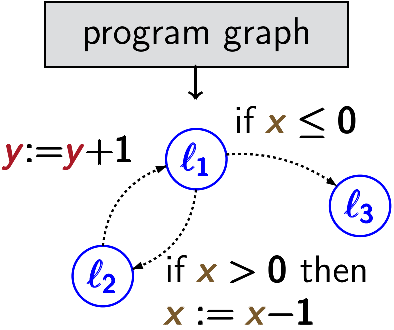
if $x \leq 0$

$\dots$

$\dots$

example: sequential program

```
WHILE  $x > 0$ DO
       $x := x-1$;
       $y := y+1$
OD
...
```



program graph

$y := y+1$

ℓ_1

if $x \leq 0$

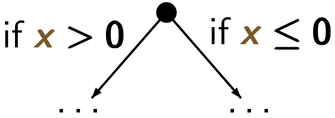
ℓ_3

ℓ_2

if $x > 0$ then $x := x-1$

Data-dependent systems

problem: TS-representation of conditional branchings **?**



if $x > 0$

if $x \leq 0$

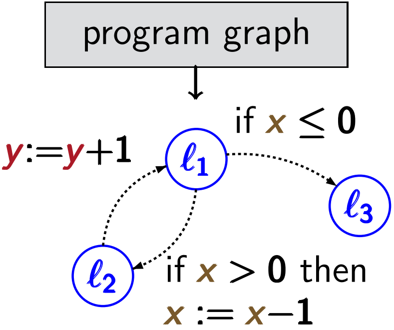
$\ldots$

$\ldots$

example: sequential program

```
$\ell_1$ →   WHILE  $x > 0$ DO
                 $x := x - 1$;
$\ell_2$ →       $y := y + 1$
             OD
$\ell_3$ →   ...
```

ℓ_1, ℓ_2, ℓ_3 are locations,
i.e., control states



program graph

$y{:=}y{+}1$

ℓ_1

if $x \leq 0$

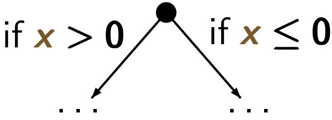
ℓ_3

ℓ_2

if $x > 0$ then
$x := x - 1$

Data-dependent systems

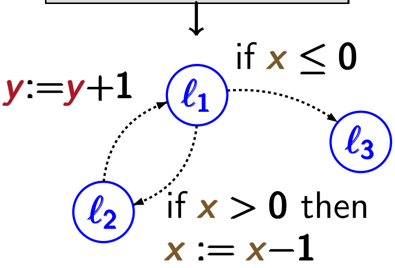
problem: TS-representation of conditional branchings **?**



example: sequential program

```
$\ell_1 \rightarrow$   WHILE  $x > 0$ DO
                 $x := x-1$;
$\ell_2 \rightarrow$             $y := y+1$
           OD
$\ell_3 \rightarrow$   $\dots$
```

program graph



states of the transition system:

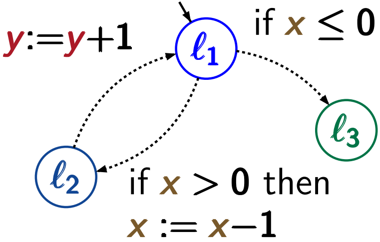
locations + relevant data (*here:* values for x and y)

Example: TS for sequential program

initially: $x = 2$, $y = 0$

```
$\ell_1 \rightarrow$  WHILE $x > 0$ DO
                $x := x-1$
$\ell_2 \rightarrow$            $y := y+1$
          OD
$\ell_3 \rightarrow$  ...
```

program graph

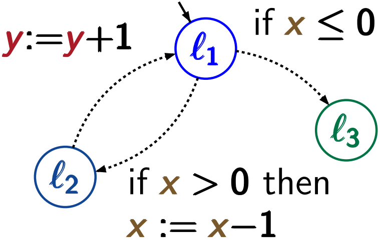


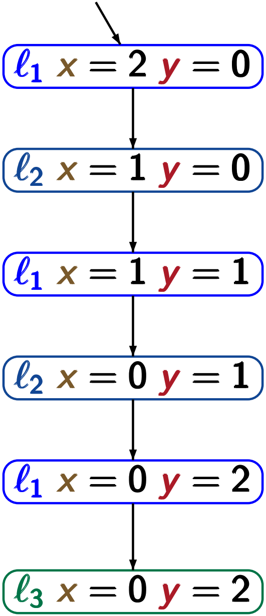
Example: TS for sequential program

initially: $x = 2$, $y = 0$

```
$\ell_1 \to$ WHILE $x > 0$ DO
                 $x := x-1$
$\ell_2 \to$     $y := y+1$
             OD
$\ell_3 \to$ ...
```

program graph





Example: TS for sequential program

initially: $x = 2$, $y = 0$

$\ell_1 \longrightarrow$ `WHILE` $x > 0$ `DO`

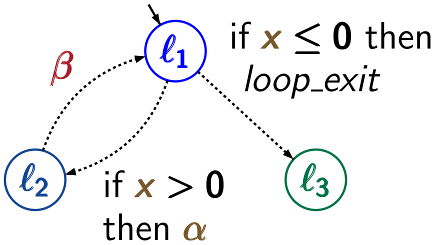
$\qquad x := x-1 \quad \longleftarrow$ $\boxed{\text{action } \alpha}$

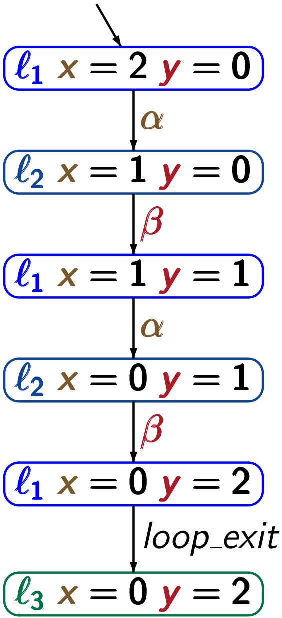
$\ell_2 \longrightarrow$ $\qquad y := y+1 \quad \longleftarrow$ $\boxed{\text{action } \beta}$

`OD`

$\ell_3 \longrightarrow$ $\ldots$

program graph





Typed variables

typed variable: variable $\boldsymbol{x}$ + data domain $\boldsymbol{Dom(x)}$

Typed variables

typed variable: variable x + data domain $Dom(x)$

- Boolean variable: variable x with $Dom(x) = \{0, 1\}$
- integer variable: variable y with $Dom(y) = \mathbb{N}$
- variable z with $Dom(z) = \{\text{yellow}, \text{red}, \text{blue}\}$

Typed variables

typed variable: variable x + data domain $Dom(x)$

- Boolean variable: variable x with $Dom(x) = \{0, 1\}$
- integer variable: variable y with $Dom(y) = \mathbb{N}$
- variable z with $Dom(z) = \{\textbf{yellow}, \textbf{red}, \textbf{blue}\}$

evaluation for a set Var of typed variables:

type-consistent function $\eta : Var \to Values$

Typed variables

typed variable: variable x + data domain $Dom(x)$

- Boolean variable: variable x with $Dom(x) = \{0, 1\}$
- integer variable: variable y with $Dom(y) = \mathbb{N}$
- variable z with $Dom(z) = \{\mathbf{yellow}, \mathbf{red}, \mathbf{blue}\}$

evaluation for a set Var of typed variables:

type-consistent function $\eta : Var \longrightarrow Values$

$\uparrow$ $\uparrow$

$\eta(x) \in Dom(x)$ for all $x \in Var$

$$Values = \bigcup_{x \in Var} Dom(x)$$

Typed variables

typed variable: variable x + data domain $Dom(x)$

- Boolean variable: variable x with $Dom(x) = \{0, 1\}$
- integer variable: variable y with $Dom(y) = \mathbb{N}$
- variable z with $Dom(z) = \{\text{yellow, red, blue}\}$

evaluation for a set Var of typed variables:

type-consistent function $\eta : Var \longrightarrow Values$

$$\eta(x) \in Dom(x) \text{ for all } x \in Var \qquad\qquad Values = \bigcup_{x \in Var} Dom(x)$$

Notation: $Eval(Var)$ = set of evaluations for Var

Conditions on typed variables

If Var is a set of typed variables then

$$\boxed{Cond(Var) = \text{ set of Boolean conditions on the variables in } Var}$$

Conditions on typed variables

If Var is a set of typed variables then

$Cond(Var) =$ set of Boolean conditions on the variables in Var

Example: $(\neg x \wedge y{<}z{+}3) \vee w{=}red$

where $Dom(x) = \{0, 1\}$, $Dom(y) = Dom(z) = \mathbb{N}$,
$Dom(w) = \{yellow, red, blue\}$

Conditions on typed variables

If Var is a set of typed variables then

$$\boxed{Cond(Var) = \text{ set of Boolean conditions on the variables in } Var}$$

Example: $\big(\neg x \wedge y{<}z{+}3\big) \vee w{=}red$

where $Dom(x) = \{0,1\}$, $Dom(y) = Dom(z) = \mathbb{N}$,
$Dom(w) = \{yellow, red, blue\}$

satisfaction relation $\models$ for evaluations and conditions

Conditions on typed variables

If Var is a set of typed variables then

$$\boxed{Cond(Var) = \text{ set of Boolean conditions on the variables in } Var}$$

Example: $(\neg x \wedge y{<}z{+}3) \vee w{=}red$

where $Dom(x) = \{0,1\}$, $Dom(y) = Dom(z) = \mathbb{N}$,
$Dom(w) = \{yellow, red, blue\}$

satisfaction relation $\models$ for evaluations and conditions

Example:

$$[x{=}0, y{=}3, z{=}6] \models \neg x \wedge y{<}z$$

$$[x{=}0, y{=}3, z{=}6] \not\models x \vee y{=}z$$

Effect-function for actions

Given a set ***Act*** of actions that operate on the variables in ***Var***, the effect of the actions is formalized by:

Effect-function for actions

Given a set Act of actions that operate on the variables in Var, the effect of the actions is formalized by:

$$Effect : Act \times Eval(Var) \longrightarrow Eval(Var)$$

Effect-function for actions

Given a set *Act* of actions that operate on the variables in *Var*, the effect of the actions is formalized by:

$$Effect : Act \times Eval(Var) \rightarrow Eval(Var)$$

if α is "$x{:=}2x+y$" then:

$$Effect(\alpha, [x{=}1, y{=}3, \ldots]) \;=\; [x{=}5, y{=}3, \ldots]$$

Effect-function for actions

Given a set *Act* of actions that operate on the variables in *Var*, the effect of the actions is formalized by:

$$\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$$

if α is "$x{:=}2x{+}y$" then:

$$\mathit{Effect}(\alpha, [x{=}1, y{=}3, \ldots]) \ = \ [x{=}5, y{=}3, \ldots]$$

if β is "$x{:=}2x{+}y \,;\, y{:=}1{-}x$" then:

$$\mathit{Effect}(\beta, [x{=}1, y{=}3, \ldots]) \ = \ [x{=}5, y{=}{-4}, \ldots]$$

Effect-function for actions

Given a set *Act* of actions that operate on the variables in *Var*, the effect of the actions is formalized by:

$$\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$$

if α is "$x{:=}2x{+}y$" then:

$$\mathit{Effect}(\alpha, [x{=}1, y{=}3, \ldots]) \;=\; [x{=}5, y{=}3, \ldots]$$

if β is "$x{:=}2x{+}y\,;\; y{:=}1{-}x$" then:

$$\mathit{Effect}(\beta, [x{=}1, y{=}3, \ldots]) \;=\; [x{=}5, y{=}{-4}, \ldots]$$

if γ is "$(x, y) := (2x{+}y, 1{-}x)$" then:

$$\mathit{Effect}(\gamma, [x{=}1, y{=}3, \ldots]) \;=\; [x{=}5, y{=}0, \ldots]$$

Program graph (PG)

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

Program graph (PG)

Let ***Var*** be a set of typed variables.

A *program graph* over ***Var*** is a tuple

$$\mathcal{P} \ = \ (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- ***Loc*** is a (finite) set of locations, i.e., control states,

Program graph (PG)

Let ***Var*** be a set of typed variables.

A *program graph* over ***Var*** is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- ***Loc*** is a (finite) set of locations, i.e., control states,
- ***Act*** a set of actions,

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$

↑

function that formalizes the effect of the actions

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$

$\uparrow$

function that formalizes the effect of the actions

example: if α is the assignment $x:=x+y$ then

$$\mathit{Effect}(\alpha, [x=1, y=7]) = [x=8, y=7]$$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \;\subseteq\; \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \; = \; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \; \subseteq \; \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  specifies conditional transitions of the form $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$

ℓ, ℓ' are locations, $g \in \mathit{Cond}(\mathit{Var})$, $\alpha \in \mathit{Act}$

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \;=\; (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \subseteq \; \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  specifies conditional transitions of the form $\ell \xrightarrow{g:\alpha} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} \ = \ (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0) \text{ where}$$

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \ \subseteq \ \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  specifies conditional transitions of the form $\ell \xrightarrow{g:\alpha} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,
- $g_0 \in \mathit{Cond}(\mathit{Var})$ initial condition on the variables

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

$$\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$$ where

- Loc is a (finite) set of locations, i.e., control states,
- Act a set of actions,
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$

  specifies conditional transitions of the form $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$
- $Loc_0 \subseteq Loc$ is the set of initial locations,
- $g_0 \in Cond(Var)$ initial condition on the variables.

TS-semantics of a program graph

TS-semantics of a program graph

program graph $\mathcal{P}$ over Var

$$\Downarrow$$

transition system $\mathcal{T}_{\mathcal{P}}$

TS-semantics of a program graph

program graph $\mathcal{P}$ over ***Var***

$$\Downarrow$$

transition system $\mathcal{T}_{\mathcal{P}}$

states in $\mathcal{T}_{\mathcal{P}}$ have the form

$\langle \ell, \eta \rangle$

location

variable evaluation

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \; = \; (S, Act, \longrightarrow, S_0, AP, L)$$

TS-semantics of a program graph

Let $\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \; = \; (S, \mathit{Act}, \longrightarrow, S_0, AP, L)$$

- state space: $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \ = \ (S, Act, \longrightarrow, S_0, AP, L)$$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \left\{ \langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0 \right\}$

TS-semantics of a program graph

Let $\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$ be a PG.
The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_{\mathcal{P}} \;=\; (S, \mathit{Act}, \longrightarrow, S_0, AP, L)$$

- state space: $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- initial states: $S_0 = \big\{ \langle \ell, \eta \rangle : \ell \in \mathit{Loc}_0, \eta \models g_0 \big\}$

The transition relation $\longrightarrow$ is given by the following rule:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \;\wedge\; \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle}$$

Structured operational semantics (SOS)

The transition system of a program graph $\mathcal{P}$ is

$$\mathcal{T}_{\mathcal{P}} \;=\; (S, Act, \longrightarrow, S_0, AP, L) \quad \text{where}$$

the transition relation $\longrightarrow$ is given by the following rule

$$\frac{\ell \xrightarrow{g:\alpha} \ell' \;\wedge\; \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

is a shortform notation in **SOS**-style.

$$\frac{premise}{conclusion}$$

Structured operational semantics (SOS)

The transition system of a program graph $\mathcal{P}$ is

$$\mathcal{T}_{\mathcal{P}} = (S, Act, \longrightarrow, S_0, AP, L) \text{ where}$$

the transition relation $\longrightarrow$ is given by the following rule

$$\frac{\ell \xhookrightarrow{g:\alpha} \ell' \;\wedge\; \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

is a shortform notation in **SOS**-style.

It means that $\longrightarrow$ is the smallest relation such that:

$$\boxed{\text{if } \ell \xhookrightarrow{g:\alpha} \ell' \;\wedge\; \eta \models g \text{ then } \langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} \ = \ (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \big\{ \langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0 \big\}$
- $\longrightarrow$ is given by the following rule:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \ \wedge \ \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

Labeling of the states

Let $\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \hookrightarrow, \mathit{Loc}_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} = (S, \mathit{Act}, \longrightarrow, S_0, \mathit{AP}, L)$

- state space: $S = \mathit{Loc} \times \mathit{Eval}(\mathit{Var})$
- initial states: $S_0 = \big\{ \langle \ell, \eta \rangle : \ell \in \mathit{Loc}_0, \eta \models g_0 \big\}$
- $\longrightarrow$ is given by the following rule:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \ \wedge \ \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \mathit{Effect}(\alpha, \eta) \rangle}$$

- atomic propositions: $\mathit{AP} = \mathit{Loc} \cup \mathit{Cond}(\mathit{Var})$

Labeling of the states

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} \ = \ (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \ \wedge \ \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

- atomic propositions: $AP = Loc \cup Cond(Var)$
- labeling function:

$$L(\langle \ell, \eta \rangle) \ = \ \{\ell\} \cup \{g \in Cond(Var) : \eta \models g\}$$

TS-semantics of a program graph

Let $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ be a PG.

transition system $\mathcal{T}_{\mathcal{P}} \; = \; (S, Act, \longrightarrow, S_0, AP, L)$

- state space: $S = Loc \times Eval(Var)$
- initial states: $S_0 = \{\langle \ell, \eta \rangle : \ell \in Loc_0, \eta \models g_0\}$
- $\longrightarrow$ is given by the following rule:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \;\wedge\; \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\eta, \alpha) \rangle}$$

- atomic propositions: $AP = Loc \cup Cond(Var)$
- labeling function:

$$L(\langle \ell, \eta \rangle) \; = \; \{\ell\} \cup \{g \in Cond(Var) : \eta \models g\}$$