Introduction

Modelling parallel systems

- Transition systems ⟵
- Modeling hard- and software systems
- Parallelism and communication

Linear Time Properties

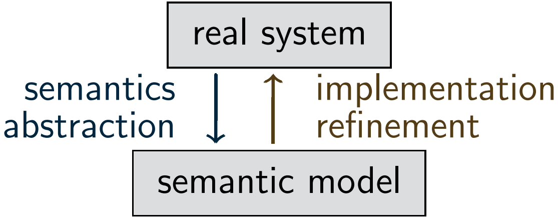
Regular Properties

Linear Temporal Logic

Computation-Tree Logic

Equivalences and Abstraction

Transition systems $\widehat{=}$ extended digraphs



The semantic model yields a formal representation of:

- the states of the system $\longleftarrow$ **nodes**
- the stepwise behaviour $\longleftarrow$ **transitions**
- the initial states
- additional information on
  communication $\longleftarrow$ **actions**
  state properties $\longleftarrow$ **atomic proposition**

Transition system (TS)

A transition system is a tuple

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

Transition system (TS)

A transition system is a tuple

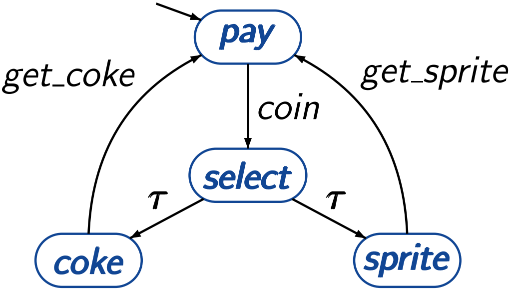
$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- S is the state space, i.e., set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is the transition relation,

  i.e., transitions have the form $s \xrightarrow{\alpha} s'$ where $s, s' \in S$ and $\alpha \in Act$

- $S_0 \subseteq S$ the set of initial states,
- AP a set of atomic propositions,
- $L : S \rightarrow 2^{AP}$ the labeling function

Transition system for beverage machine

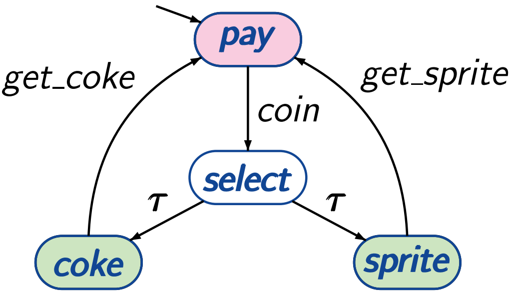


actions:
coin
τ
get_sprite
get_coke

state space $S = \{pay, select, coke, sprite\}$

set of initial states: $S_0 = \{pay\}$

Transition system for beverage machine



actions:
coin
τ
get_sprite
get_coke

state space $S = \{pay, select, coke, sprite\}$

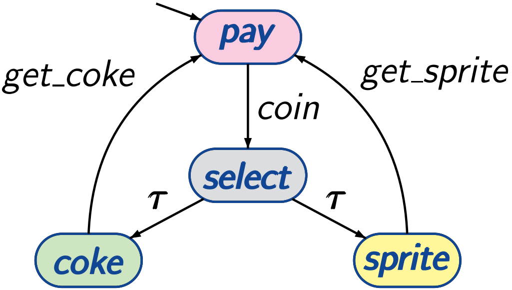
set of initial states: $S_0 = \{pay\}$

set of atomic propositions: $AP = \{pay, drink\}$

labeling function: $L(coke) = L(sprite) = \{drink\}$

$L(pay) = \{pay\}$, $L(select) = \emptyset$

Transition system for beverage machine



actions:
coin
τ
get_sprite
get_coke

state space $S = \{pay, select, coke, sprite\}$

set of initial states: $S_0 = \{pay\}$

set of atomic propositions: $AP = S$

labeling function: $L(s) = \{s\}$ for each state s

"Behaviour" of transition systems

possible behaviours of a TS result from:

```
select nondeterministically an initial state s ∈ S_0
WHILE  s is non-terminal DO
      select nondeterministically a transition s --α--> s'
      execute the action α and put s := s'
OD
```

executions: maximal "transition sequences"

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots \text{ with } s_0 \in S_0$$

reachable fragment:

$Reach(\mathcal{T})$ = set of all states that are reachable from an initial state through some execution

Transition system for parallel actions

parallel execution of independent actions

$$\text{e.g.}\ \underbrace{x := x+1}_{\text{action } \alpha} \ ||| \ \underbrace{y := y-3}_{\text{action } \beta} \qquad \alpha, \beta \text{ independent}$$

parallel execution of dependent actions

$$\text{e.g.}\ \underbrace{x := x+1}_{\text{action } \alpha} \ ||| \ \underbrace{y := 2*x}_{\text{action } \beta} \qquad \alpha, \beta \text{ dependent}$$

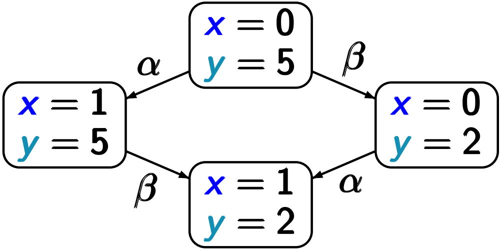
Transition system for parallel actions

parallel execution of independent actions ⟵ interleaving

e.g. $\underbrace{x := x{+}1}_{\text{action } \alpha} \;|||\; \underbrace{y := y{-}3}_{\text{action } \beta}$ α, β independent

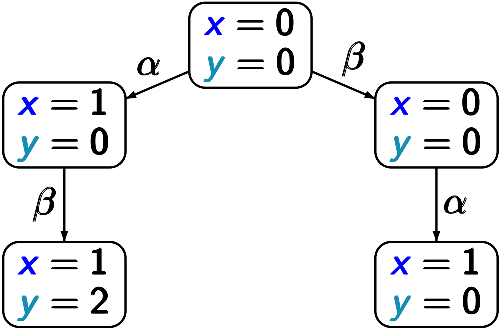
parallel execution of dependent actions ⟵ competition

e.g. $\underbrace{x := x{+}1}_{\text{action } \alpha} \;|||\; \underbrace{y := 2{*}x}_{\text{action } \beta}$ α, β dependent parallel execution of independent actions ⟵ interleaving



$$\underbrace{x := x{+}1}_{\text{action } \alpha} \;|||\; \underbrace{y := y{-}3}_{\text{action } \beta}$$

parallel execution of dependent actions ⟵ competition



$$\underbrace{x := x{+}1}_{\text{action } \alpha} \;|||\; \underbrace{y := 2{*}x}_{\text{action } \beta}$$

Introduction

Modelling parallel systems

- Transition systems
- Modeling hard- and software systems ⟵
- Parallelism and communication

Linear Time Properties

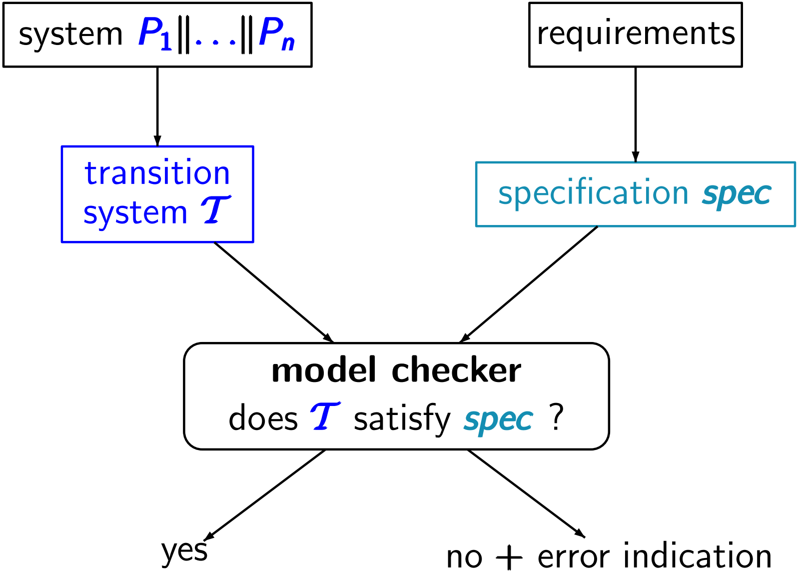
Regular Properties

Linear Temporal Logic

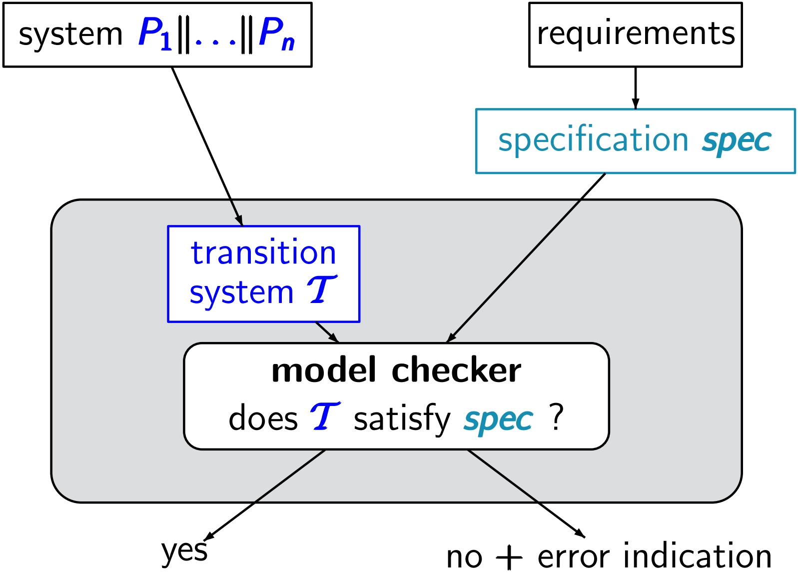
Computation-Tree Logic

Equivalences and Abstraction

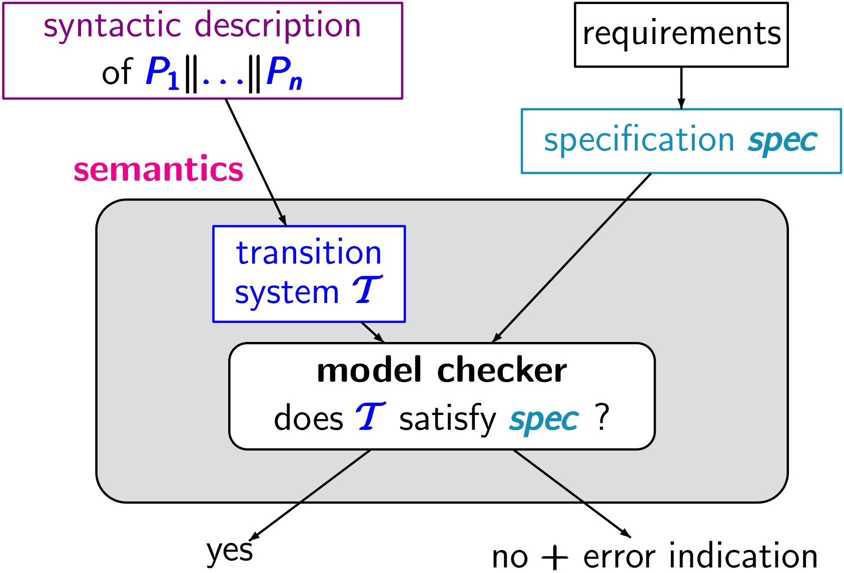
Model checking



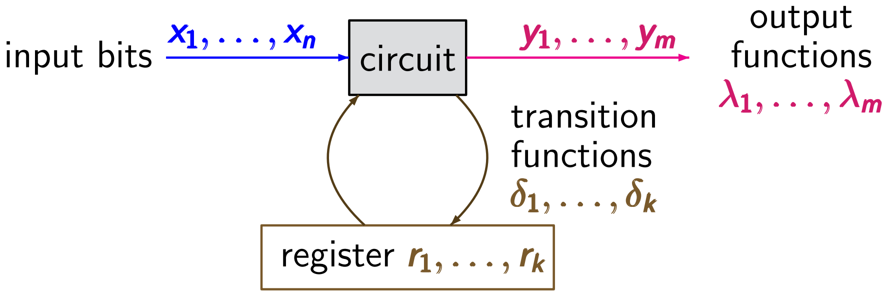
Model checking



Model checking



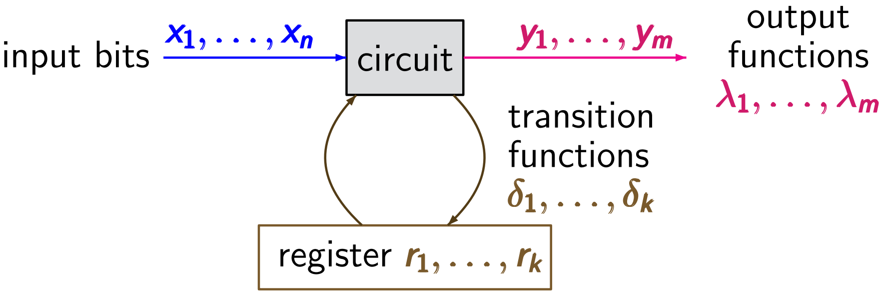
Modelling of sequential circuits by TS



$$\delta_j, \lambda_i \mathrel{\widehat{=}} \text{switching functions } \{0,1\}^n \times \{0,1\}^k \longrightarrow \{0,1\}$$

input values $a_1, \ldots, a_n$ for the input variables		output value $\lambda_i(\ldots)$ for output variable y_i
+ current values $c_1, \ldots, c_k$ of the registers	$\mapsto$	next value $\delta_j(\ldots)$ for register r_j

Modelling of sequential circuits by TS

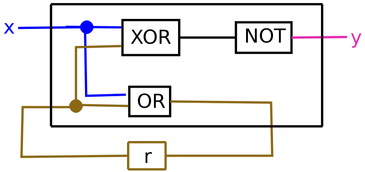


initial register evaluation $[r_1{=}c_{01}, \ldots, r_k{=}c_{0k}]$

transition system:

- states: evaluations of $x_1, \ldots, x_n, r_1, \ldots, r_k$
- transitions represent the stepwise behavior
- values of input bits change nondeterministically
- atomic propositions: $x_1, \ldots, x_n, y_1, \ldots, y_m, r_1, \ldots, r_k$

Example: TS for sequential circuit



output function

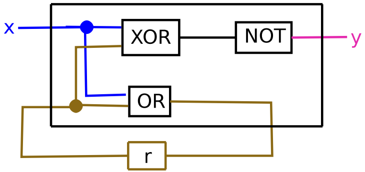
$\lambda_y = \neg(x \oplus r)$

transition function

$\delta_r = x \vee r$

transition system

Example: TS for sequential circuit



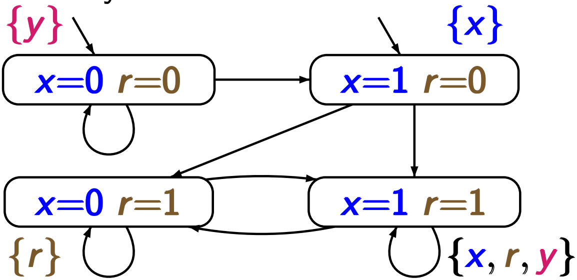
output function

$\lambda_y = \neg(x \oplus r)$

transition function

$\delta_r = x \vee r$

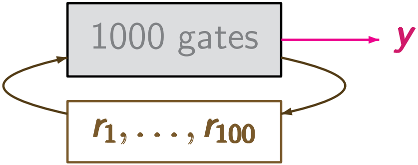
transition system



initial register evaluation: r=0

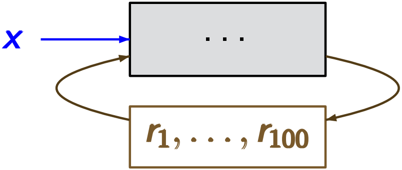
How many states . . .

. . . has the transition system for a circuit of the form?



1 output bit
no input
100 registers

answer: 2^{100}

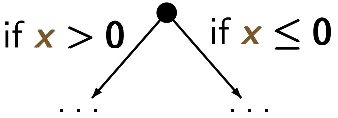


no output
1 input bit
100 registers

answer: $2^{100} * 2^1 = 2^{101}$

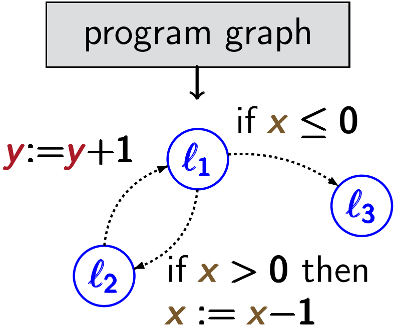
Data-dependent systems

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



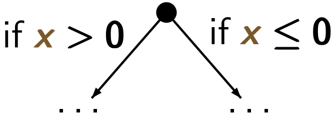
example: sequential program

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



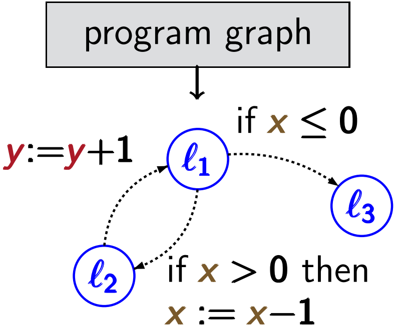
ℓ_1, ℓ_2, ℓ_3 are locations,
i.e., control states

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



example: sequential program

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



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

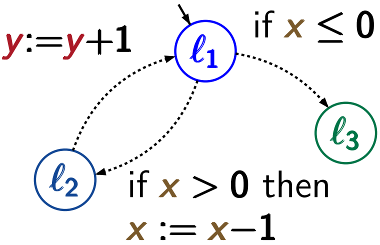
$\quad\quad x := x-1$

$\ell_2 \rightarrow$ $\quad y := y+1$

OD

$\ell_3 \rightarrow$ ...

program graph



Example: TS for sequential program

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

$\ell_1 \rightarrow$ WHILE $x > 0$ DO

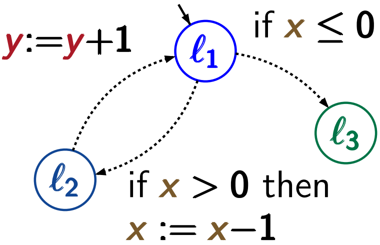
$\qquad x := x - 1$

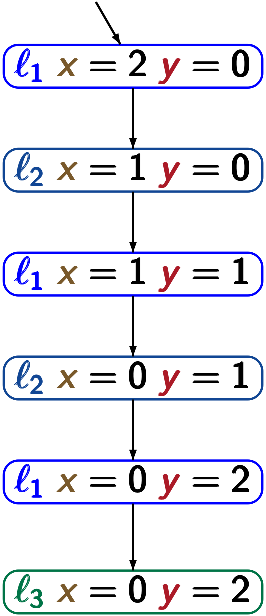
$\ell_2 \rightarrow \qquad y := y + 1$

OD

$\ell_3 \rightarrow \ldots$

program graph





Example: TS for sequential program

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

$\ell_1 \rightarrow$ WHILE $x > 0$ DO

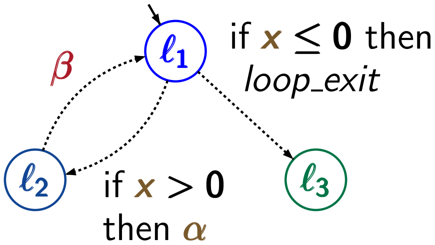
$x := x-1$ $\longleftarrow$ action α

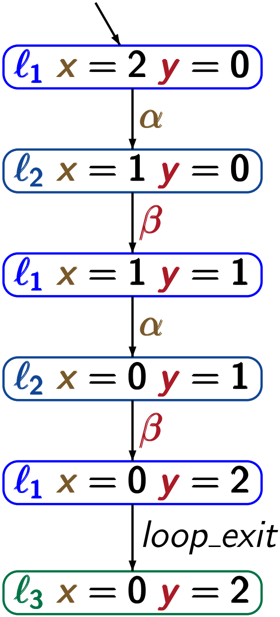
$\ell_2 \rightarrow$ $y := y+1$ $\longleftarrow$ action β

OD

$\ell_3 \rightarrow$ ...

program graph





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 $\eta : Var \to Values$

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

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

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

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

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:

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

if α is "$x := 2x + y$" then:

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

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

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

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

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

Program graph (PG)

Let Var be a set of typed variables.

A *program graph* over Var is a tuple

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

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

↑

function that formalizes the effect of the actions

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

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

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,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

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

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,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$

  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 $\ell \overset{g:\alpha}{\hookrightarrow} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,
- $g_0 \in \mathit{Cond}(\mathit{Var})$ initial condition on the variables.

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:

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

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

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

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

Structured operational semantics (SOS)

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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