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 $\cong$ extended digraphs

The semantic model yields a formal representation of:

- the states of the system $\leftarrow$ nodes
- the stepwise behaviour $\leftarrow$ transitions
- the initial states
- additional information on
  - communication $\leftarrow$ actions
  - state properties $\leftarrow$ atomic proposition
A transition system is a tuple

\[ T = (S, \text{Act}, \rightarrow, S_0, AP, L) \]
A transition system is a tuple

$$T = (S, Act, \rightarrow, S_0, AP, L)$$

- $S$ is the state space, i.e., set of states,
A transition system is a tuple

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

- $S$ is the state space, i.e., set of states,
- $Act$ is a set of actions,
A transition system is a tuple

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

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

\[ T = (S, Act, \rightarrow, S_0, AP, L) \]

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

i.e., transitions have the form \( s \xrightarrow{\alpha} s' \)

where \( s, s' \in S \) and \( \alpha \in Act \)
Transition system (TS)

A transition system is a tuple

\[ T = (S, \text{Act}, \rightarrow, S_0, \text{AP}, L) \]

- \( S \) is the state space, i.e., set of states,
- \( \text{Act} \) is a set of actions,
- \( \rightarrow \subseteq S \times \text{Act} \times S \) is the transition relation,
- i.e., transitions have the form \( s \overset{\alpha}{\rightarrow} s' \) where \( s, s' \in S \) and \( \alpha \in \text{Act} \),
- \( S_0 \subseteq S \) the set of initial states,
A transition system is a tuple

\[ T = (S, \text{Act}, \rightarrow, S_0, AP, L) \]

- \( S \) is the state space, i.e., set of states,
- \( \text{Act} \) is a set of actions,
- \( \rightarrow \subseteq S \times \text{Act} \times S \) is the transition relation,
- i.e., transitions have the form \( s \xrightarrow{\alpha} s' \) where \( s, s' \in S \) and \( \alpha \in \text{Act} \)
- \( S_0 \subseteq S \) the set of initial states,
- \( AP \) a set of atomic propositions,
- \( L : S \rightarrow 2^{AP} \) the labeling function
Transition system for beverage machine
Transition system for beverage machine

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

Set of initial states: $S_0 = \{\text{pay}\}$
Transition system for beverage machine

actions:
- coin
- get_sprite
- get_coke

state space \( S = \{ \text{pay, select, coke, sprite} \} \)

set of initial states: \( S_0 = \{ \text{pay} \} \)
Transition system for beverage machine

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

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

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

Labeling function:
- $L(coke) = L(sprite) = \{drink\}$
- $L(pay) = \{pay\}$
- $L(select) = \emptyset$

Actions:
- coin
- $\tau$
- get_sprite
- get_coke
Transition system for beverage machine

State space \( S = \{ \text{pay}, \text{select}, \text{coke}, \text{sprite} \} \)

Set of initial states: \( S_0 = \{ \text{pay} \} \)

Set of atomic propositions: \( AP = S \)

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

Actions:
- coin
- \( \tau \)
- get_sprite
- get_coke

Diagram:
- Start at pay
- Transition to get_coke and get_sprite
- Transition from get_coke to pay
- Transition from get_sprite to select
- Transition from select back to get_coke and get_sprite
“Behaviour” of transition systems

possible behaviours of a TS result from:

```
select nondeterministically an initial state \( s \in S_0 \)
WHILE \( s \) is non-terminal DO

    select nondeterministically a transition \( s \xrightarrow{\alpha} s' \)
    execute the action \( \alpha \) and put \( s := s' \)

OD
```
“Behaviour” of transition systems

possible behaviours of a TS result from:

select nondeterministically an initial state $s \in S_0$

WHILE $s$ is non-terminal DO

    select nondeterministically a transition $s \xrightarrow{\alpha} s'$

    execute the action $\alpha$ and put $s := s'$

OD

executions: maximal “transition sequences”

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots \text{ with } s_0 \in S_0$$
possible behaviours of a TS result from:

```
select nondeterministically an initial state \( s \in S_0 \)
WHILE \( s \) is non-terminal DO
    select nondeterministically a transition \( s \xrightarrow{\alpha} s' \)
    execute the action \( \alpha \) 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:

\[
\text{Reach}(T) = \text{set of all states that are reachable from an initial state through some execution}
\]
Transition system for parallel actions

parallel execution of independent actions

parallel execution of dependent actions
Transition system for parallel actions

parallel execution of independent actions

e.g. \( x := x + 1 \parallel y := y - 3 \) \( \alpha, \beta \) independent

parallel execution of dependent actions
Transition system for parallel actions

parallel execution of independent actions

e.g. \( x := x + 1 \) \( \| \| \) \( y := y - 3 \)  \( \alpha, \beta \) independent

\begin{align*}
\text{action } \alpha \\
\text{action } \beta
\end{align*}

parallel execution of dependent actions

e.g. \( x := x + 1 \) \( \| \| \) \( y := 2 \times x \) \( \alpha, \beta \) dependent

\begin{align*}
\text{action } \alpha \\
\text{action } \beta
\end{align*}
Transition system for parallel actions

parallel execution of independent actions ← interleaving

e.g. $\begin{align} x &:= x + 1 \\
\text{action } \alpha \end{align}$ $\parallel$ $\begin{align} y &:= y - 3 \\
\text{action } \beta \end{align}$ $\alpha$, $\beta$ independent

parallel execution of dependent actions ← competition

e.g. $\begin{align} x &:= x + 1 \\
\text{action } \alpha \end{align}$ $\parallel$ $\begin{align} y &:= 2 \times x \\
\text{action } \beta \end{align}$ $\alpha$, $\beta$ dependent
parallel execution of independent actions ← interleaving

\[
\begin{align*}
\text{action } \alpha & : x := x + 1 \\
\text{action } \beta & : y := y - 3
\end{align*}
\]
parallel execution of independent actions \rightarrow \text{interleaving}

parallel execution of dependent actions \rightarrow \text{competition}

\begin{align*}
\alpha & : x = 1, y = 5 \\
\beta & : x = 0, y = 5 \\
\alpha & : x = 1, y = 2 \\
\beta & : x = 0, y = 2 \\
\text{action } \alpha & : x := x + 1 \\
\text{action } \beta & : y := y - 3
\end{align*}
parallel execution of independent actions

\[
\begin{align*}
\alpha &: x = 0, y = 5 \\
\beta &: x = 1, y = 2
\end{align*}
\]

action \( \alpha \) \( x := x + 1 \) \( || \) action \( \beta \) \( y := y - 3 \)

parallel execution of dependent actions

\[
\begin{align*}
\alpha &: x = 0, y = 0 \\
\beta &: x = 1, y = 2
\end{align*}
\]

action \( \alpha \) \( x := x + 1 \) \( || \) action \( \beta \) \( y := 2 \times x \)
Overview

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

system $P_1 \parallel \ldots \parallel P_n$

transition system $\mathcal{T}$

requirements

specification $spec$

model checker

does $\mathcal{T}$ satisfy $spec$?

yes

no + error indication
Model checking

system $P_1 || \ldots || P_n$

requirements

transition system $\mathcal{T}$

model checker

does $\mathcal{T}$ satisfy spec ?

yes

no + error indication
Model checking

syntactic description of \( P_1 \| \ldots \| P_n \)

requirements

specification \( spec \)

transition system \( \mathcal{T} \)

model checker

does \( \mathcal{T} \) satisfy \( spec \) ?

semantics

yes

no + error indication
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ \rightarrow circuit \rightarrow output bits $y_1, \ldots, y_m$

register $r_1, \ldots, r_k$
Modelling of sequential circuits by TS

Input bits $x_1, \ldots, x_n$ to circuit, which has register $r_1, \ldots, r_k$. Outputs $y_1, \ldots, y_m$ are determined by output functions $\lambda_1, \ldots, \lambda_m$. Transition functions $\delta_1, \ldots, \delta_k$ are involved in the operation of the circuit.
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ \rightarrow circuit \rightarrow output functions $y_1, \ldots, y_m$

register $r_1, \ldots, r_k$

transition functions $\delta_1, \ldots, \delta_k$

$\delta_j, \lambda_i \equiv$ switching functions $\{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}$
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ \rightarrow circuit \rightarrow output functions $y_1, \ldots, y_m$

transition functions $\delta_1, \ldots, \delta_k$

register $r_1, \ldots, r_k$

$\delta_j, \lambda_i \equiv$ switching functions $\{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}$

input values $a_1, \ldots, a_n$ for the input variables

$+$ current values $c_1, \ldots, c_k$ of the registers

output value $\lambda_i(\ldots)$ for output variable $y_i$

next value $\delta_j(\ldots)$ for register $r_j$
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ → circuit → transition functions $\delta_1, \ldots, \delta_k$ → register $r_1, \ldots, r_k$ → output functions $\lambda_1, \ldots, \lambda_m$

initial register evaluation $[r_1 = c_{01}, \ldots, r_k = c_{0k}]$
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ \arrow{circuit} \arrow{output functions $y_1, \ldots, y_m$}

register $r_1, \ldots, r_k$

transition functions $\delta_1, \ldots, \delta_k$

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

transition system:

- states: evaluations of $x_1, \ldots, x_n, r_1, \ldots, r_k$
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ \rightarrow circuit \rightarrow output functions $y_1, \ldots, y_m$

register $r_1, \ldots, r_k$

transition functions $\delta_1, \ldots, \delta_k$

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
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ -> circuit -> output functions $y_1, \ldots, y_m$

transition functions $\delta_1, \ldots, \delta_k$

register $r_1, \ldots, r_k$

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
Modelling of sequential circuits by TS

input bits $x_1, \ldots, x_n$ → circuit

transition functions $\delta_1, \ldots, \delta_k$

output functions $\lambda_1, \ldots, \lambda_m$

register $r_1, \ldots, r_k$

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: sequential circuit
Example: sequential circuit

output function: \[ \lambda_y = \neg(x \oplus r) \]

transition function: \[ \delta_r = x \lor r \]
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]
Example: TS for sequential circuit

\[ \lambda_y = \neg(x \oplus r) \]

\[ \delta_r = x \lor r \]

transition system

<table>
<thead>
<tr>
<th>x=0 ( r=0 )</th>
<th>x=1 ( r=0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0 ( r=1 )</td>
<td>x=1 ( r=1 )</td>
</tr>
</tbody>
</table>
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]

transition system

\hline
<table>
<thead>
<tr>
<th>State</th>
<th>x=0</th>
<th>r=0</th>
<th>x=1</th>
<th>r=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>(x=0)</td>
<td>(r=0)</td>
<td>(x=1)</td>
<td>(r=0)</td>
</tr>
<tr>
<td></td>
<td>(x=0)</td>
<td>(r=1)</td>
<td>(x=1)</td>
<td>(r=1)</td>
</tr>
</tbody>
</table>
\hline

initial register evaluation: \(r=0\)
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]

transition system

initial register evaluation: \( r=0 \)
Example: TS for sequential circuit

输出函数：
$$\lambda_y = \neg(x \oplus r)$$

转换函数：
$$\delta_r = x \lor r$$

转换系统

初始寄存器评估：$$r=0$$
Example: TS for sequential circuit

Output function
\[ \lambda_y = \neg (x \oplus r) \]

Transition function
\[ \delta_r = x \lor r \]

Transition system

Initial register evaluation: \( r=0 \)
Example: TS for sequential circuit

\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]

transition system

initial register evaluation: \( r=0 \)
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg (x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]

circuit diagram

transition system

\[
\begin{array}{c}
\text{x=0 r=0} \\
\text{x=1 r=0} \\
\text{x=0 r=1} \\
\text{x=1 r=1}
\end{array}
\]

initial register evaluation: \( r=0 \)
Example: TS for sequential circuit

output function
\[ \lambda_y = \neg(x \oplus r) \]

transition function
\[ \delta_r = x \lor r \]

transition system
\[
\begin{align*}
\{y\} & \quad \{x\} \\
\{r\} & \quad \{r\}
\end{align*}
\]

initial register evaluation: \( r=0 \)
Example: TS for sequential circuit

\[ \lambda_y = \neg (x \oplus r) \]

\[ \delta_r = x \lor r \]

transition system

\{y\} \rightarrow x=0 \; r=0 \rightarrow x=1 \; r=0 \rightarrow x=0 \; r=1 \rightarrow \{r\} \rightarrow \{x\}

\{x\} \rightarrow x=1 \; r=0 \rightarrow x=1 \; r=1 \rightarrow \{x, r, y\} \rightarrow \{x\}

initial register evaluation: \( r=0 \)
How many states ... 

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

1000 gates

$r_1, \ldots, r_{100}$

$y$

1 output bit
no input
100 registers
How many states ... has the transition system for a circuit of the form?

answer: \(2^{100}\)
How many states ... has the transition system for a circuit of the form?

1000 gates

\( r_1, \ldots, r_{100} \)

\( y \)

answer: \( 2^{100} \)

\( x \)

\( r_1, \ldots, r_{100} \)

1 output bit
no input
100 registers

1 input bit
no output
100 registers
How many states . . .

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

1000 gates

$r_1, \ldots, r_{100}$

1 output bit
no input
100 registers

Answer: $2^{100}$

$x$

$\ldots$

$r_1, \ldots, r_{100}$

no output
1 input bit
100 registers

Answer: $2^{100} \times 2^1 = 2^{101}$
Data-dependent systems

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

\[
\text{if } x > 0 \quad \text{if } x \leq 0
\]
Data-dependent systems

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

\[
\text{if } x > 0 \quad \bullet \quad \text{if } x \leq 0
\]

\[
\ldots \quad \ldots
\]

**example:** sequential program

\[
\text{WHILE } x > 0 \text{ DO}
\]
\[
x := x - 1;
\]
\[
y := y + 1
\]
\[
\text{OD}
\]
\[
\ldots
\]
**Data-dependent systems**

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

\[ \text{if } x > 0 \quad \text{if } x \leq 0 \]

\[ \ldots \quad \ldots \]

**Example:** sequential program

\[
\text{WHILE } x > 0 \text{ DO} \\
\quad x := x - 1; \\
\quad y := y + 1 \\
\text{OD} \\
\ldots
\]

\[
\text{if } x \leq 0 \quad \text{if } x > 0 \text{ then} \\
\quad y := y + 1 \\
\quad \text{if } x > 0 \text{ then} \\
\quad x := x - 1
\]
Data-dependent systems

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

```plaintext
if x > 0  
  . . .
if x ≤ 0  
  . . .
```

**Example:** sequential program

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

Program graph:

- Initial state: $l_1$
- Condition: $x > 0$
- Action: $y := y + 1$
- Next state: $l_2$

- Initial state: $l_2$
- Condition: $x ≤ 0$
- Action: $y := y + 1$
- Next state: $l_1$

- Initial state: $l_1$
- Condition: $x > 0$
- Action: $x := x - 1$
- Next state: $l_3$

- Initial state: $l_3$
- Condition: $x ≤ 0$
- Action: $x := x - 1$
- Next state: $l_2$
Data-dependent systems

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

\[
\begin{align*}
\text{if } x > 0 & \quad & \text{if } x \leq 0 \\
\cdots & \quad & \cdots
\end{align*}
\]

**example:** sequential program

\[
\begin{align*}
\ell_1 \rightarrow & \quad \text{WHILE } x > 0 \text{ DO} \\
& \quad x := x - 1; \\
\ell_2 \rightarrow & \quad y := y + 1 \\
\ell_3 \rightarrow & \quad \text{OD}
\end{align*}
\]

\[
\begin{align*}
\ell_1, \ell_2, \ell_3 \text{ are locations,} \\
i.e., \text{ control states}
\end{align*}
\]
Data-dependent systems

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

- if $x > 0$
- if $x \leq 0$

**example:** sequential program

$l_1 \rightarrow$ WHILE $x > 0$ DO
  $x := x-1$;

$l_2 \rightarrow$ y := y+1

$l_3 \rightarrow$ OD

states of the transition system:

locations $+$ relevant data  (*here:* values for $x$ and $y$)
Example: TS for sequential program

initially: $x = 2, y = 0$

$l_1 \rightarrow$ WHILE $x > 0$ DO

\hspace{1cm} $x := x - 1$

$l_2 \rightarrow$ y := y+1

OD

$l_3 \rightarrow$ ...

program graph

$y := y + 1 \quad l_1$ if $x \leq 0$

$l_2$ if $x > 0$ then

\hspace{1cm} $x := x - 1$

$l_3$
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

\( \ell_3 \rightarrow \ldots \)

program graph

\( y := y + 1 \) if \( x \leq 0 \)

\( \ell_2 \) if \( x > 0 \) then
  \( x := x - 1 \)
Example: TS for sequential program

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

$l_1$ → WHILE $x > 0$ DO

$x := x - 1$ ← action $\alpha$

$l_2$ →

$y := y + 1$ ← action $\beta$

OD

$l_3$ → ...

program graph

if $x \leq 0$ then loop_exit

if $x > 0$ then $\alpha$

$l_2$ → $\beta$

$l_3$ → $\beta$

$l_1 x = 2$ $y = 0$

$l_2 x = 1$ $y = 0$

$l_1 x = 1$ $y = 1$

$l_2 x = 0$ $y = 1$

$l_1 x = 0$ $y = 2$

$l_3 x = 0$ $y = 2$

loop_exit
Typed variables

*typed variable:* variable $x$ + data domain $\text{Dom}(x)$
Typed variables

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

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

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

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

evaluation for a set $\text{Var}$ of typed variables:

  type-consistent function $\eta : \text{Var} \rightarrow \text{Values}$
Typed variables

**typed variable**: variable \( x \) + data domain \( \text{Dom}(x) \)

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

**evaluation** for a set \( \text{Var} \) of typed variables:

- type-consistent function \( \eta : \text{Var} \rightarrow \text{Values} \)
  
  \[ \eta(x) \in \text{Dom}(x) \]
  
  for all \( x \in \text{Var} \)
Typed variables

Typed variable: variable \( x \) + data domain \( \text{Dom}(x) \)

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

evaluation for a set \( \text{Var} \) of typed variables:

type-consistent function \( \eta : \text{Var} \rightarrow \text{Values} \)

\[
\eta(x) \in \text{Dom}(x)
\]

for all \( x \in \text{Var} \)

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

Notation: \( \text{Eval}(\text{Var}) = \) set of evaluations for \( \text{Var} \)
Conditions on typed variables

If $\text{Var}$ is a set of typed variables then

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

If \( \text{Var} \) is a set of typed variables then

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

Example: \((\neg x \land y < z + 3) \lor w = \text{red}\)

where \( \text{Dom}(x) = \{0, 1\} \), \( \text{Dom}(y) = \text{Dom}(z) = \mathbb{N} \),
\( \text{Dom}(w) = \{\text{yellow, red, blue}\} \)
If $\text{Var}$ is a set of typed variables then

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

Example: $$(\neg x \land y < z + 3) \lor w = \text{red}$$

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

$satisfaction\ relation$ $\models$ for evaluations and conditions
Conditions on typed variables

If \( \textbf{Var} \) is a set of typed variables then

\[
\text{Cond}(\textbf{Var}) = \text{set of Boolean conditions on the variables in } \textbf{Var}
\]

Example: \( (\neg x \land y < z + 3) \lor w = \text{red} \)

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

satisfaction relation \( \models \) for evaluations and conditions

Example:
\[
[x=0, y=3, z=6] \models \neg x \land y < z
\]
\[
[x=0, y=3, z=6] \not\models x \lor y = z
\]
Effect-function for actions

Given a set \( \text{Act} \) of actions that operate on the variables in \( \text{Var} \), the effect of the actions is formalized by:
Given a set $\mathit{Act}$ of actions that operate on the variables in $\mathit{Var}$, the effect of the actions is formalized by:

$$\text{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \to \mathit{Eval}(\mathit{Var})$$
Effect-function for actions

Given a set \( \text{Act} \) of actions that operate on the variables in \( \text{Var} \), the effect of the actions is formalized by:

\[
\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})
\]

if \( \alpha \) is “\( x := 2x + y \)” then:

\[
\text{Effect}(\alpha, [x=1, y=3, \ldots]) = [x=5, y=3, \ldots]
\]
Effect-function for actions

Given a set \( \text{Act} \) of actions that operate on the variables in \( \text{Var} \), the effect of the actions is formalized by:

\[
\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})
\]

if \( \alpha \) is “\( x\,\text{:=}\,2x+y \)” then:

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

if \( \beta \) is “\( x\,\text{:=}\,2x+y \); \( y\,\text{:=}\,1-x \)” then:

\[
\text{Effect}(\beta, [x=1, y=3, \ldots]) = [x=5, y=-4, \ldots]
\]
Effect-function for actions

Given a set \( \text{Act} \) of actions that operate on the variables in \( \text{Var} \), the effect of the actions is formalized by:

\[
\text{Effect} : \text{Act} \times \text{Eval(Var)} \rightarrow \text{Eval(Var)}
\]

if \( \alpha \) is “\( x:=2x+y \)” then:

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

if \( \beta \) is “\( x:=2x+y ; y:=1-x \)” then:

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

if \( \gamma \) is “\( (x, y) := (2x+y, 1-x) \)” then:

\[
\text{Effect}(\gamma, [x=1, y=3, \ldots]) = [x=5, y=0, \ldots]
\]
Program graph (PG)
Program graph (PG)

Let \( \text{Var} \) be a set of typed variables.

A program graph over \( \text{Var} \) is a tuple

\[
\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)
\]

where
Let $\text{Var}$ be a set of typed variables.

A program graph over $\text{Var}$ is a tuple

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

where

- $\text{Loc}$ is a (finite) set of locations, i.e., control states,
Let \( \textbf{Var} \) be a set of typed variables.

A program graph over \( \textbf{Var} \) is a tuple

\[
\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)
\]

where

- \( \text{Loc} \) is a (finite) set of locations, i.e., control states,
- \( \text{Act} \) a set of actions,
Let $\text{Var}$ be a set of typed variables.

A program graph over $\text{Var}$ is a tuple

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

where

- $\text{Loc}$ is a (finite) set of locations, i.e., control states,
- $\text{Act}$ a set of actions,
- $\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})$
Let $\text{Var}$ be a set of typed variables.

A program graph over $\text{Var}$ is a tuple

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

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

function that formalizes the effect of the actions
Let \( \text{Var} \) be a set of typed variables.

A program graph over \( \text{Var} \) is a tuple

\[
\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)
\]

where

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

function that formalizes the effect of the actions

example: if \( \alpha \) is the assignment \( x := x + y \) then

\[
\text{Effect}(\alpha, [x=1, y=7]) = [x=8, y=7]
\]
Let $\text{Var}$ be a set of typed variables.

A program graph over $\text{Var}$ is a tuple

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

where

- $\text{Loc}$ is a (finite) set of locations, i.e., control states,
- $\text{Act}$ a set of actions,
- $\text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var})$,
- $\rightarrow \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times \text{Act} \times \text{Loc}$
Let \( \textit{Var} \) be a set of typed variables.

A \textit{program graph} over \( \textit{Var} \) is a tuple

\[
\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)
\]

where

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

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

\( \ell, \ell' \) are locations, \( g \in \text{Cond}(\text{Var}) \), \( \alpha \in \text{Act} \)
Let $\mathit{Var}$ be a set of typed variables.

A program graph over $\mathit{Var}$ is a tuple

$$
\mathcal{P} = (\mathit{Loc}, \mathit{Act}, \mathit{Effect}, \rightarrow, \mathit{Loc}_0, \mathit{g}_0)
$$

where

- $\mathit{Loc}$ is a (finite) set of locations, i.e., control states,
- $\mathit{Act}$ a set of actions,
- $\mathit{Effect} : \mathit{Act} \times \mathit{Eval}(\mathit{Var}) \rightarrow \mathit{Eval}(\mathit{Var})$
- $\rightarrow \subseteq \mathit{Loc} \times \mathit{Cond}(\mathit{Var}) \times \mathit{Act} \times \mathit{Loc}$ specifies conditional transitions of the form $\ell \xleftarrow{\mathit{g} : \mathit{\alpha}} \ell'$
- $\mathit{Loc}_0 \subseteq \mathit{Loc}$ is the set of initial locations,
Program graph (PG)

Let $\mathbf{Var}$ be a set of typed variables.

A program graph over $\mathbf{Var}$ is a tuple

$$\mathcal{P} = (\mathbf{Loc}, \mathbf{Act}, \mathbf{Effect}, \rightarrow, \mathbf{Loc}_0, g_0)$$

where

- $\mathbf{Loc}$ is a (finite) set of locations, i.e., control states,
- $\mathbf{Act}$ a set of actions,
- $\mathbf{Effect} : \mathbf{Act} \times \mathbf{Eval}(\mathbf{Var}) \rightarrow \mathbf{Eval}(\mathbf{Var})$,
- $\rightarrow \subseteq \mathbf{Loc} \times \mathbf{Cond}(\mathbf{Var}) \times \mathbf{Act} \times \mathbf{Loc}$
  specifies conditional transitions of the form $\ell \xrightarrow{\text{\scriptsize \texttt{g:}\alpha}} \ell'$
- $\mathbf{Loc}_0 \subseteq \mathbf{Loc}$ is the set of initial locations,
- $g_0 \in \mathbf{Cond}(\mathbf{Var})$ initial condition on the variables.
Let \( \text{Var} \) be a set of typed variables.

A program graph over \( \text{Var} \) is a tuple

\[
\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)
\]

where

- \( \text{Loc} \) is a (finite) set of locations, i.e., control states,
- \( \text{Act} \) a set of actions,
- \( \text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var}) \)
- \( \rightarrow \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times \text{Act} \times \text{Loc} \)
  specifies conditional transitions of the form \( \ell \xleftarrow{g : \alpha} \ell' \)
- \( \text{Loc}_0 \subseteq \text{Loc} \) is the set of initial locations,
- \( g_0 \in \text{Cond}(\text{Var}) \) initial condition on the variables.
TS-semantics of a program graph
TS-semantics of a program graph

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

$\Downarrow$

transition system $\mathcal{T}_\mathcal{P}$
TS-semantics of a program graph

Program graph $\mathcal{P}$ over $\text{Var}$

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

States in $\mathcal{T}_\mathcal{P}$ have the form $\langle l, \eta \rangle$

- Location
- Variable evaluation
Let $\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \leadsto, \text{Loc}_0, g_0)$ be a PG.

The transition system of $\mathcal{P}$ is:

$$\mathcal{T}_\mathcal{P} = (S, \text{Act}, \rightarrow, S_0, AP, L)$$
TS-semantics of a program graph

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

The transition system of $\mathcal{P}$ is:

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

- state space: $S = \text{Loc} \times \text{Eval}(\text{Var})$
Let $\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)$ be a PG. The transition system of $\mathcal{P}$ is:

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

- state space: $S = \text{Loc} \times \text{Eval}(\text{Var})$
- initial states: $S_0 = \{ (l, \eta) : l \in \text{Loc}_0, \eta \models g_0 \}$
Let $\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \twoheadrightarrow, \text{Loc}_0, g_0)$ be a PG.

The transition system of $\mathcal{P}$ is:

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

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

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

$$\frac{l \xleftarrow{g:\alpha} l' \land \eta \models g}{\langle l, \eta \rangle \xrightarrow{\alpha} \langle l', \text{Effect}(\alpha, \eta) \rangle}$$
Structured operational semantics (SOS)

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

\[
\mathcal{I}_\mathcal{P} = (S, \text{Act}, \rightarrow, S_0, AP, L)
\]

where the transition relation \( \rightarrow \) is given by the following rule

\[
\alpha \vdash \ell \rightarrow \ell' \land \eta \models g \\
\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle
\]

is a shortform notation in SOS-style.
Structured operational semantics (SOS)

The transition system of a program graph $P$ is

$$
\mathcal{I}_P = (S, Act, \rightarrow, S_0, AP, L)
$$

where

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

$$
\ell \xrightarrow{\alpha} \ell' \land \eta \models g
\rightarrow
\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle
$$

is a shortform notation in SOS-style.

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

if $\ell \xrightarrow{\alpha} \ell' \land \eta \models g$ then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$
Let \( \mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \vdash, \text{Loc}_0, g_0) \) be a PG.

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

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

\[
\ell \xleftarrow{g: \alpha} \ell' \land \eta \models g
\]

\[
\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle
\]
Labeling of the states

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

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

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

\[
\ell \xleftarrow{g:\alpha} \ell' \land \eta \models g \quad \Rightarrow \quad \langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle
\]

- atomic propositions: $\text{AP} = \text{Loc} \cup \text{Cond}(\text{Var})$
Labeling of the states

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

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

• state space: $S = \text{Loc} \times \text{Eval}(\text{Var})$

• initial states: $S_0 = \{ \langle \ell, \eta \rangle : \ell \in \text{Loc}_0, \eta \models g_0 \}$

• $\rightarrow$ is given by the following rule:

\[
\ell \xrightleftharpoons[\alpha \vdash \text{Effect}(\alpha, \eta)]{g : \alpha} \ell' \land \eta \models g
\]

$\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$

• atomic propositions: $\text{AP} = \text{Loc} \cup \text{Cond}(\text{Var})$

• labeling function:

$L(\langle \ell, \eta \rangle) = \{ \ell \} \cup \{ g \in \text{Cond}(\text{Var}) : \eta \models g \}$
Let $\mathcal{P} = (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0)$ be a PG.

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

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

$$\ell \xleftarrow{g: \alpha} \ell' \land \eta \models g$$

$$\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect} (\eta, \alpha) \rangle$$

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

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

by Dijkstra
Guarded Command Language (GCL)

by Dijkstra

- high-level modeling language that contains features of imperative languages and nondeterministic choice
Guarded Command Language (GCL) by Dijkstra

- high-level modeling language that contains features of imperative languages and nondeterministic choice

- semantics:

```
GCL-program
↓↓↓
program graph
↓↓↓
transition system
```
guarded command $g \Rightarrow stmt$

$g$ : guard, i.e., Boolean condition on the program variables

$stmt$ : statement
Guarded Command Language (GCL)

guarded command \( g \Rightarrow stmt \) ← enabled if \( g \) is true

- \( g \) : guard, i.e., Boolean condition on the program variables
- \( stmt \) : statement
Guarded Command Language (GCL)

**guarded command** $g \Rightarrow stmt$  \hspace{1cm} \leftarrow \text{enabled if } g \text{ is true}

\begin{align*}
g & : \text{guard, i.e., Boolean condition on the program variables} \\
stmt & : \text{statement}
\end{align*}

**repetitive command/loop:**

```
DO :: g \Rightarrow stmt OD
```
guarded command $g \Rightarrow stmt$ ← enabled if $g$ is true

$g$ : guard, i.e., Boolean condition on the program variables

$stmt$ : statement

repetitive command/loop:

DO :: $g \Rightarrow stmt$ OD ← WHILE $g$ DO $stmt$ OD
Guarded Command Language (GCL)

guarded command \( g \rightarrow stmt \) \(\xrightarrow{\small{\text{enabled if } g \text{ is true}}}\)

\[\begin{array}{l}
g : \text{guard, i.e., Boolean condition on the program variables} \\
stmt : \text{statement}
\end{array}\]

repetitive command/loop:

\[
\begin{array}{c}
\text{DO} \; : \; g \rightarrow stmt \; \text{OD} \\
\text{WHILE} \; g \; \text{DO} \; stmt \; \text{OD}
\end{array}
\]

conditional command:

\[
\begin{array}{c}
\text{IF} \; : \; g \rightarrow stmt_1 \\
\quad : \; \neg g \rightarrow stmt_2 \\
\text{FI}
\end{array}
\]
Guarded Command Language (GCL)

guarded command $g \Rightarrow stmt$ ← enabled if $g$ is true

- $g$ : guard, i.e., Boolean condition on the program variables
- $stmt$ : statement

repetitive command/loop:

DO :: $g \Rightarrow stmt$ OD ← WHILE $g$ DO $stmt$ OD

conditional command:

IF :: $g \Rightarrow stmt_1$

:: $\neg g \Rightarrow stmt_2$

FI

IF $g$ THEN $stmt_1$

ELSE $stmt_2$

FI
guarded command $g \Rightarrow stmt$ ← enabled if $g$ is true

repetitive command/loop:

DO :: $g \Rightarrow stmt$ OD ← WHILE $g$ DO $stmt$ OD

conditional command:

IF :: $g \Rightarrow stmt_1$
:: $\neg g \Rightarrow stmt_2$
FI

IF $g$ THEN $stmt_1$
ELSE $stmt_2$
FI

symbol :: stands for the nondeterministic choice between enabled guarded commands
Guarded Command Language (GCL)

modeling language with nondeterministic choice

\[
\begin{align*}
\text{stmt} & \quad \text{def} \quad x := \text{expr} \quad | \quad \text{stmt}_1; \text{stmt}_2 \quad | \\
\text{DO} & \quad ::g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad ::g_n \Rightarrow \text{stmt}_n \quad \text{OD} \\
\text{IF} & \quad ::g_1 \Rightarrow \text{stmt}_1 \quad \ldots \quad ::g_n \Rightarrow \text{stmt}_n \quad \text{FI} \\
\vdots
\end{align*}
\]

where \(x\) is a typed variable and \(\text{expr}\) an expression of the same type
Guarded Command Language (GCL)

modeling language with nondeterministic choice

<table>
<thead>
<tr>
<th>stmt</th>
<th>def</th>
<th>x := expr</th>
<th>stmt₁; stmt₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DO :: g₁ ⇒ stmt₁ ... :: gₙ ⇒ stmtₙ OD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IF :: g₁ ⇒ stmt₁ ... :: gₙ ⇒ stmtₙ FI</td>
<td></td>
</tr>
</tbody>
</table>

where \( x \) is a typed variable and \( expr \) an expression of the same type

semantics of a GCL-program: program graph
GCL-program for beverage machine
uses two variables $\#\text{sprite}, \#\text{coke} \in \{0, 1, \ldots, \text{max}\}$ for the number of available drinks (sprite or coke)
uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, \text{max}\}$ for the number of available drinks (sprite or coke)

uses the following actions:

<table>
<thead>
<tr>
<th></th>
<th>enabled</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_coke</td>
<td>if $#coke &gt; 0$</td>
<td>$#coke := #coke - 1$</td>
</tr>
<tr>
<td>get_sprite</td>
<td>if $#sprite &gt; 0$</td>
<td>$#sprite := #sprite - 1$</td>
</tr>
</tbody>
</table>
GCL-program for beverage machine

uses two variables \(#\text{sprite}, \#\text{coke} \in \{0, 1, \ldots, \text{max}\}\)
for the number of available drinks (sprite or coke)

uses the following actions:

<table>
<thead>
<tr>
<th></th>
<th>enabled</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_coke</td>
<td>if #\text{coke} &gt; 0</td>
<td>#\text{coke} := #\text{coke} − 1</td>
</tr>
<tr>
<td>get_sprite</td>
<td>if #\text{sprite} &gt; 0</td>
<td>#\text{sprite} := #\text{sprite} − 1</td>
</tr>
<tr>
<td>refill</td>
<td>any time</td>
<td>#\text{sprite} := \text{max} &lt;br&gt;#\text{coke} := \text{max}</td>
</tr>
</tbody>
</table>
uses two variables $\#\text{sprite}, \#\text{coke} \in \{0, 1, \ldots, \max\}$ for the number of available drinks (sprite or coke)

uses the following actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>enabled</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_coke</td>
<td>if $#\text{coke} &gt; 0$</td>
<td>$#\text{coke} := #\text{coke} - 1$</td>
</tr>
<tr>
<td>get_sprite</td>
<td>if $#\text{sprite} &gt; 0$</td>
<td>$#\text{sprite} := #\text{sprite} - 1$</td>
</tr>
<tr>
<td>refill</td>
<td>any time</td>
<td>$#\text{sprite} := \max$  $#\text{coke} := \max$</td>
</tr>
<tr>
<td>insert_coin</td>
<td>any time</td>
<td>no effect on variables</td>
</tr>
</tbody>
</table>
uses two variables $\#\text{sprite}, \#\text{coke} \in \{0,1,\ldots,\text{max}\}$ for the number of available drinks (sprite or coke)

uses the following actions:

<table>
<thead>
<tr>
<th></th>
<th>enabled</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_coke</td>
<td>if $#\text{coke} &gt; 0$</td>
<td>$#\text{coke} := #\text{coke} - 1$</td>
</tr>
<tr>
<td>get_sprite</td>
<td>if $#\text{sprite} &gt; 0$</td>
<td>$#\text{sprite} := #\text{sprite} - 1$</td>
</tr>
</tbody>
</table>
| refill           | any time    | $\#\text{sprite} := \text{max}$  
                              | $\#\text{coke} := \text{max}$         |
| insert_coin      | any time    | no effect on variables   |
| return_coin      | if machine is empty and user has entered a coin (no effect on variables) | |
GCL-program for beverage machine

DO :: true ⇒ insert_coin;

IF :: #sprite = #coke = 0 ⇒ return_coin

:: #coke > 0 ⇒ #coke := #coke − 1

:: #sprite > 0 ⇒ #sprite := #sprite − 1

FI

:: true ⇒ #sprite := max; #coke := max

OD
GCL-program for beverage machine

DO :: true ⇒ insert_coin; (* user inserts a coin *)

  IF :: #sprite = #coke = 0 ⇒ return_coin

     :: #coke > 0 ⇒ #coke := #coke − 1

        :: #sprite > 0 ⇒ #sprite := #sprite − 1

  FI

  :: true ⇒ #sprite := max; #coke := max

OD
GCL-program for beverage machine

DO :: true ⇒ insert_coin; (* user inserts a coin *)

IF :: \#sprite = \#coke = 0 ⇒ return_coin
(* no beverage available *)

:: \#coke > 0 ⇒ \#coke := \#coke − 1

:: \#sprite > 0 ⇒ \#sprite := \#sprite − 1

FI

:: true ⇒ \#sprite := max; \#coke := max

OD
GCL-program for beverage machine

DO :: true ⇒ insert_coin; (* user inserts a coin *)

IF :: #sprite = #coke = 0 ⇒ return_coin
    (* no beverage available *)
    :: #coke > 0 ⇒ #coke := #coke − 1
        (* user selects coke *)
    :: #sprite > 0 ⇒ #sprite := #sprite−1
        (* user selects sprite *)
FI

:: true ⇒ #sprite := max; #coke := max
    (* refilling of the machine *)
OD
GCL-program for beverage machine

DO :: true ⇒ insert_coin; (* user inserts a coin *)

IF :: #sprite = #coke = 0 ⇒ return_coin
(* no beverage available *)

:: #coke > 0 ⇒ get_coke
(* user selects coke *)

:: #sprite > 0 ⇒ get<Sprite>
(* user selects sprite *)

FI

:: true ⇒ refill
(* refilling of the machine *)

OD
GCL-program for beverage machine

DO :: true ⇒ insert_coin;

IF :: #sprite = #coke = 0 ⇒ return_coin

:: #coke > 0 ⇒ get_coke

:: #sprite > 0 ⇒ get_sprite

FI

:: true ⇒ refill

OD
GCL-program for beverage machine

\[
\begin{align*}
\text{DO} & \quad \text{true} \Rightarrow \text{insert\_coin}; \\
\text{IF} & \quad \#\text{sprite} = \#\text{coke} = 0 \\
& \quad \Rightarrow \text{return\_coin} \\
& \quad \#\text{coke} > 0 \Rightarrow \text{get\_coke} \\
& \quad \#\text{sprite} > 0 \Rightarrow \text{get\_sprite} \\
\text{FI} \\
\text{DO} & \quad \text{true} \Rightarrow \text{refill}
\end{align*}
\]

... yields a program graph with

- two variables \#\text{sprite}, \#\text{coke} \in \{0, 1, \ldots, \text{max}\}
GCL-program for beverage machine

\texttt{\textbf{start} → DO :: true ⇒ insert\_coin;}

\texttt{\textbf{select}→}

\texttt{IF :: \#sprite = \#coke = 0}

\texttt{⇒ return\_coin}

\texttt{:: \#coke > 0 ⇒ get\_coke}

\texttt{:: \#sprite > 0 ⇒ get\_sprite}

\texttt{FI}

\texttt{:: true ⇒ refill}

\texttt{OD}

... yields a program graph with

- two variables \#\texttt{sprite}, \#\texttt{coke} ∈ \{0, 1, \ldots, max\}
- two locations \texttt{start} and \texttt{select}