Overview

Transition systems $\widehat{=}$ extended digraphs

real system

semantics, abstraction ↓ ↑ implementation, refinement

semantic model

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

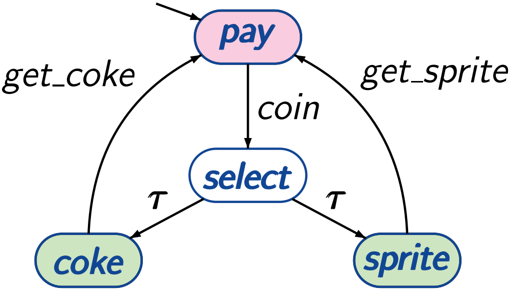
$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- 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,
- $L : S \to 2^{AP}$ the labeling function

Transition system for beverage machine



actions:
- *coin*
- τ
- *get_sprite*
- *get_coke*

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

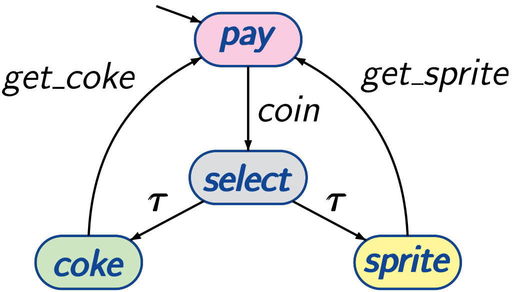
set of initial states: $S_0 = \{pay\}$

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 ∈ S0
WHILE  s is non-terminal DO
      select nondeterministically a transition s --α--> s'
      execute the action α and put s := s'
OD
```

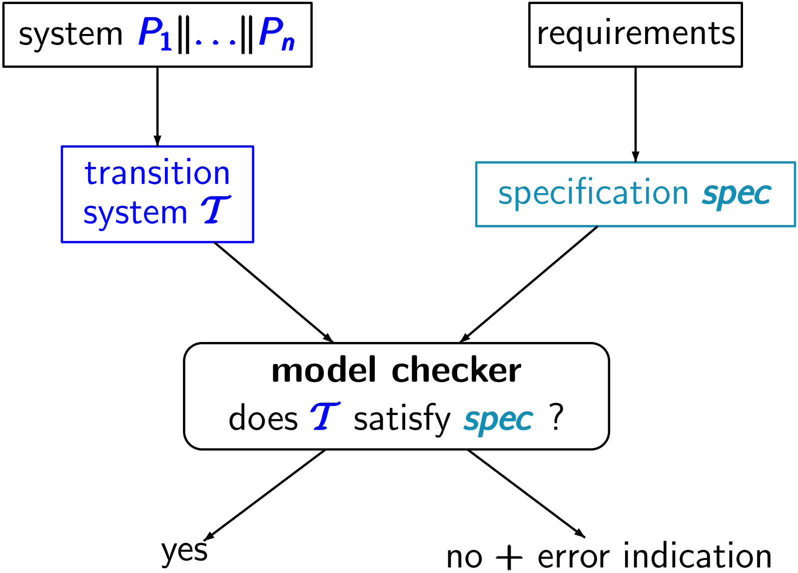
executions: maximal "transition sequences"

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

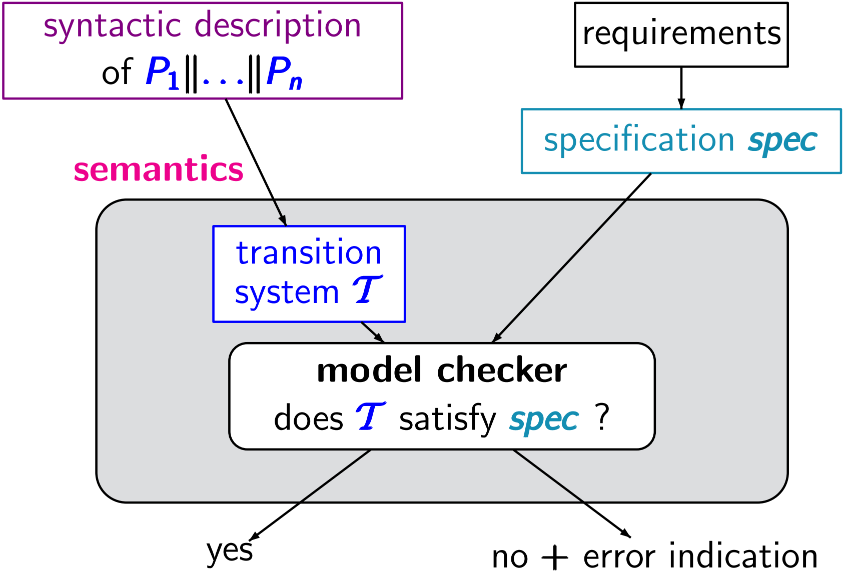
reachable fragment:

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

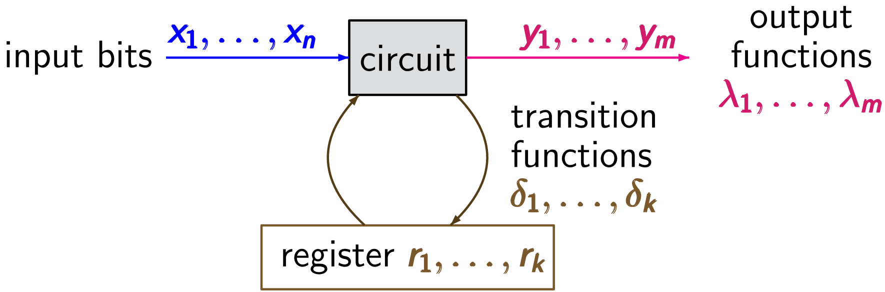
Model checking



Model checking



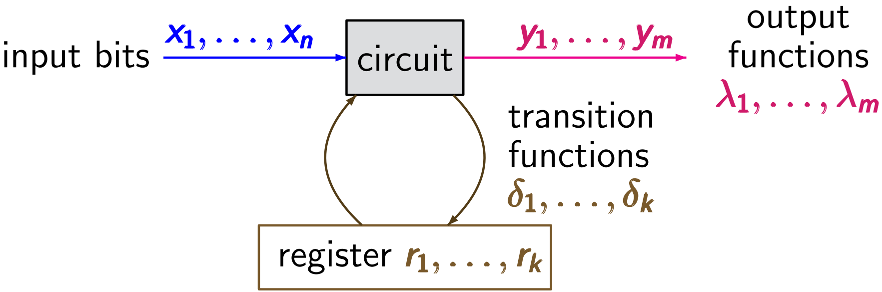
Modelling of sequential circuits by TS



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

input values $a_1, \ldots, a_n$ for the input variables $+$ current values $c_1, \ldots, c_k$ of the registers	$\mapsto$	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

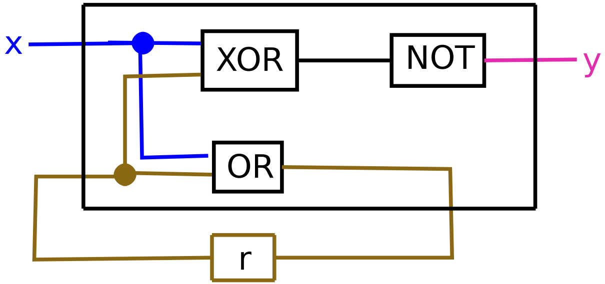


initial register evaluation $[r_1 = c_{01}, \ldots, r_k = c_{0k}]$

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: $x_1, \ldots, x_n, y_1, \ldots, y_m, r_1, \ldots, r_k$

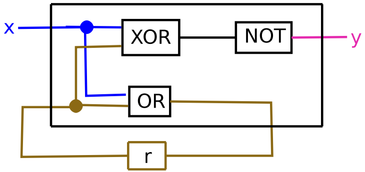
Example: sequential circuit



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

transition function: $\delta_r = x \vee r$

Example: TS for sequential circuit



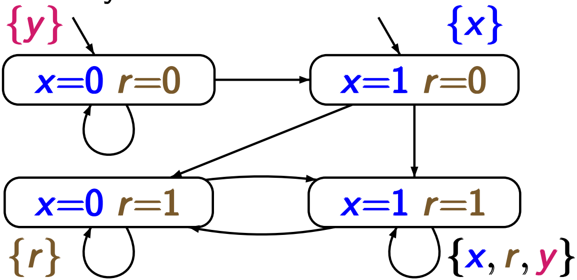
output function

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

transition function

$\delta_r = x \vee r$

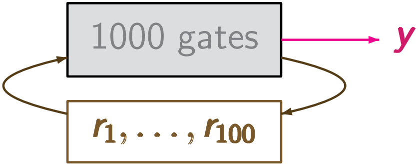
transition system



initial register evaluation: r=0

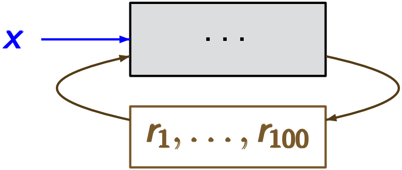
How many states . . .

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



1 output bit
no input
100 registers

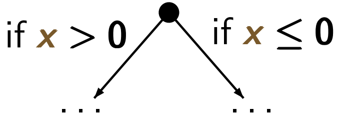
answer: 2^{100}



no output
1 input bit
100 registers

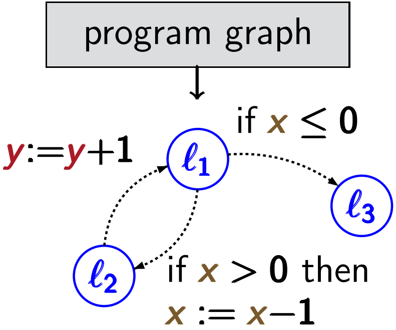
answer: $2^{100} * 2^1 = 2^{101}$

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



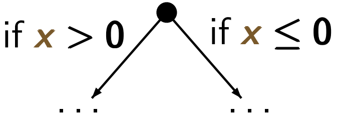
example: sequential program

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



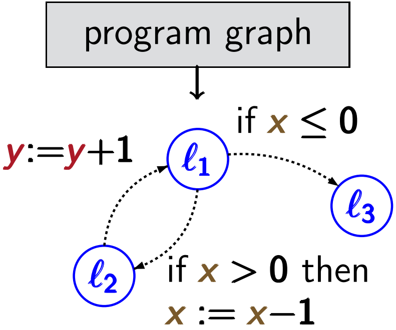
ℓ_1, ℓ_2, ℓ_3 are locations,
i.e., control states

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



example: sequential program

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



states of the transition system:

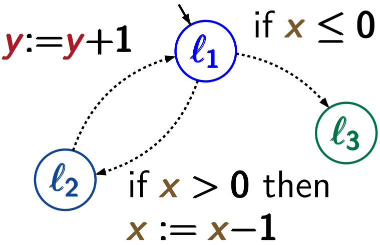
locations + relevant data (*here:* values for x and y)

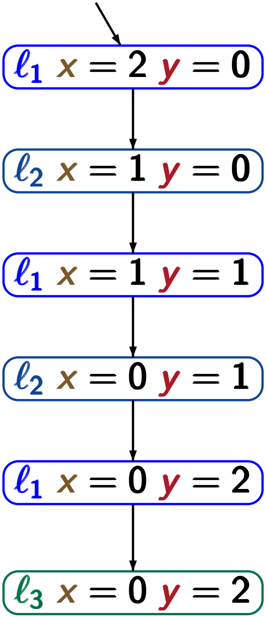
Example: TS for sequential program

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

```
ℓ1→   WHILE  x > 0 DO
              x := x−1
ℓ2→           y := y+1
      OD
ℓ3→   ...
```

program graph





Example: TS for sequential program

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

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

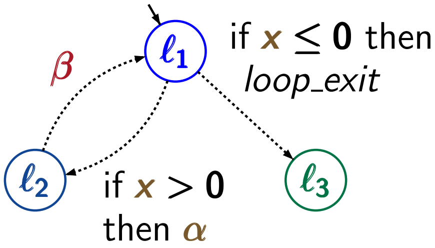
$\qquad x := x - 1 \quad \longleftarrow$ $\boxed{\text{action } \alpha}$

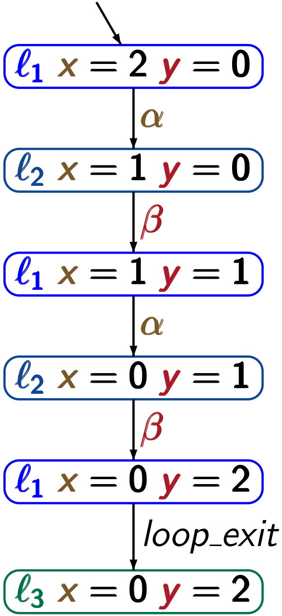
$\ell_2 \rightarrow \qquad y := y + 1 \quad \longleftarrow$ $\boxed{\text{action } \beta}$

`OD`

$\ell_3 \rightarrow \ \ldots$

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 $\eta : Var \rightarrow Values$

$\eta(x) \in Dom(x)$ 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

$$\boxed{\mathit{Cond}(\mathit{Var}) = \begin{array}{l}\text{set of Boolean conditions}\\ \text{on the variables in } \mathit{Var}\end{array}}$$

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

TS-semantics of a program graph

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:

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

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

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

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

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

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

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

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

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

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

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:

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

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 Language (GCL)

guarded command $g \Rightarrow \textit{stmt}$ ⟵ enabled if g is true

g	:	guard, i.e., Boolean condition on the program variables
$\textit{stmt}$	:	statement

repetitive command/loop:

DO $:: g \Rightarrow \textit{stmt}$ OD ⟵ WHILE g DO $\textit{stmt}$ OD

conditional command:

IF $:: \; g \Rightarrow \textit{stmt}_1$
$:: \; \neg g \Rightarrow \textit{stmt}_2$
FI

⟵

IF g THEN $\textit{stmt}_1$
ELSE $\textit{stmt}_2$
FI

Guarded Command Language (GCL)

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

repetitive command/loop:

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

conditional command:

```
IF  ::  g ⇒ stmt1
    :: ¬g ⇒ stmt2
FI
```

$\longleftarrow$

```
IF g THEN stmt1
     ELSE stmt2
FI
```

symbol :: stands for the nondeterministic choice between enabled guarded commands

Guarded Command Language (GCL)

modeling language with nondeterministic choice

$$
\begin{array}{lll}
\textit{stmt} & \stackrel{\text{def}}{=} & x := \textit{expr} \quad \big| \quad \textit{stmt}_1 ; \textit{stmt}_2 \quad \big| \\
 & & \texttt{DO} :: g_1 \Rightarrow \textit{stmt}_1 \;\; \ldots \;\; :: g_n \Rightarrow \textit{stmt}_n \;\texttt{OD} \\
 & & \texttt{IF} :: g_1 \Rightarrow \textit{stmt}_1 \;\; \ldots \;\; :: g_n \Rightarrow \textit{stmt}_n \;\texttt{FI} \\
 & & \vdots
\end{array}
$$

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 $\#\mathit{sprite}, \#\mathit{coke} \in \{0, 1, \ldots, \mathit{max}\}$ for the number of available drinks (sprite or coke)

uses the following actions:

	enabled	effect
get_coke	if $\#\mathit{coke} > 0$	$\#\mathit{coke} := \#\mathit{coke} - 1$
get_sprite	if $\#\mathit{sprite} > 0$	$\#\mathit{sprite} := \#\mathit{sprite} - 1$
refill	any time	$\#\mathit{sprite} := \mathit{max}$ $\#\mathit{coke} := \mathit{max}$
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

$$
\begin{array}{lll}
\texttt{DO} & :: & \text{true} \Rightarrow \text{insert\_coin};\ (\text{* user inserts a coin *}) \\
& & \quad \texttt{IF} \ :: \ \#sprite = \#coke = 0 \Rightarrow \text{return\_coin} \\
& & \qquad\qquad\qquad\qquad (\text{* no beverage available *}) \\
& & \quad \phantom{\texttt{IF}} \ :: \ \#coke > 0 \Rightarrow \#coke := \#coke - 1 \\
& & \qquad\qquad\qquad\qquad (\text{* user selects coke *}) \\
& & \quad \phantom{\texttt{IF}} \ :: \ \#sprite > 0 \Rightarrow \#sprite := \#sprite - 1 \\
& & \qquad\qquad\qquad\qquad (\text{* user selects sprite *}) \\
& & \quad \texttt{FI} \\
& :: & \text{true} \Rightarrow \#sprite := max; \#coke := max \\
& & \qquad\qquad\qquad\qquad (\text{* refilling of the machine *}) \\
\texttt{OD} & &
\end{array}
$$

GCL-program for beverage machine

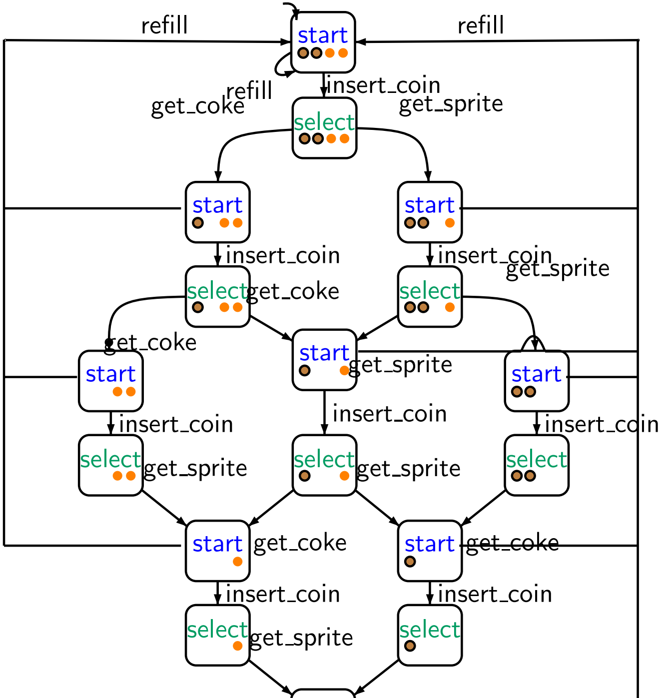

```
DO :: true ⇒ insert_coin; (* user inserts a coin *)
      IF :: #sprite = #coke = 0 ⇒ return_coin
                                 (* no beverage available *)
         :: #coke > 0 ⇒ get_coke
                                 (* user selects coke *)
         :: #sprite > 0 ⇒ get_sprite
                                 (* user selects sprite *)
      FI
   :: true ⇒ refill
                                 (* refilling of the machine *)
OD
```

GCL-program for beverage machine

$$
\begin{array}{llll}
\mathit{start} \longrightarrow \texttt{DO} & :: & \mathrm{true} \Rightarrow \mathrm{insert\_coin}; & \\
\mathit{select} \longrightarrow & & \texttt{IF} \; :: \; \#\mathit{sprite} = \#\mathit{coke} = 0 & \\
& & \qquad\qquad\qquad \Rightarrow \mathrm{return\_coin} & \\
& & \quad\;\; :: \; \#\mathit{coke} > 0 \Rightarrow \mathrm{get\_coke} & \\
& & \quad\;\; :: \; \#\mathit{sprite} > 0 \Rightarrow \mathrm{get\_sprite} & \\
& & \texttt{FI} & \\
& :: & \mathrm{true} \Rightarrow \mathrm{refill} & \\
\texttt{OD} & & &
\end{array}
$$

... yields a program graph with

- two variables $\#\mathit{sprite}, \#\mathit{coke} \in \{0, 1, \ldots, \mathit{max}\}$
- two locations *start* and *select*


refill
start
refill
insert_coin
refill
get_coke
get_sprite
select
start
start
insert_coin
insert_coin
get_sprite
select
get_coke
select
get_coke
start
start
get_sprite
start
insert_coin
insert_coin
insert_coin
select
get_sprite
select
get_sprite
select
start
get_coke
start
get_coke
insert_coin
insert_coin
select
get_sprite
select