Finite-State Machine (FSM)
A.K.A. Finite-State Automaton

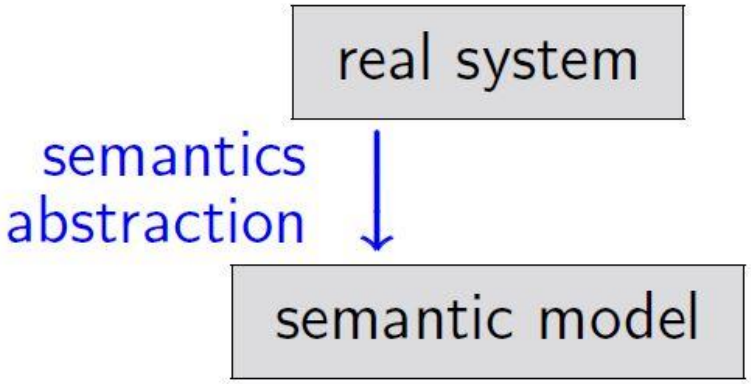
- *Mathematical model of computation* used to design both computer programs and sequential logic circuits
- Abstract machine (M) can be in one of a finite number of *states*
- M is in only one sate at a time; the state it is in at any given time is called the *current* state
- Can change from one state to another when initiated by a triggering event or conditions; this is called *transition*

State and Transition (concepts and terminology)

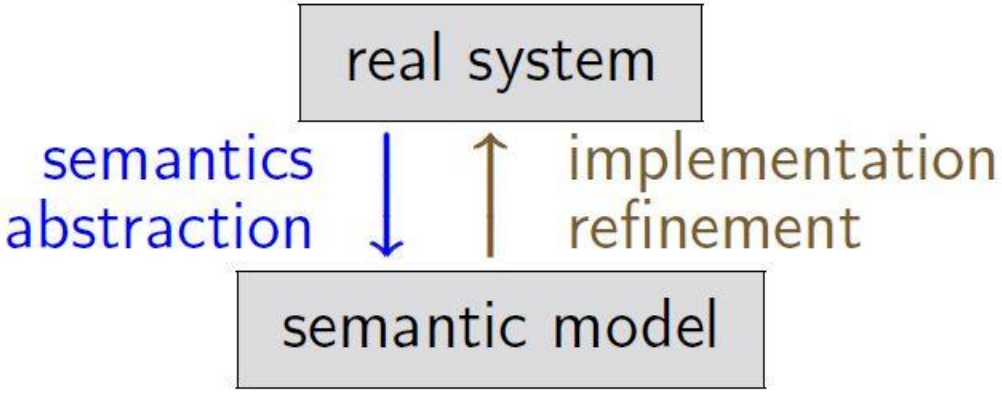
- *A state is a description of the status of M that is waiting to execute a transition*

- *A transition is a set of actions to be executed when a condition is fulfilled or when an event is received.*

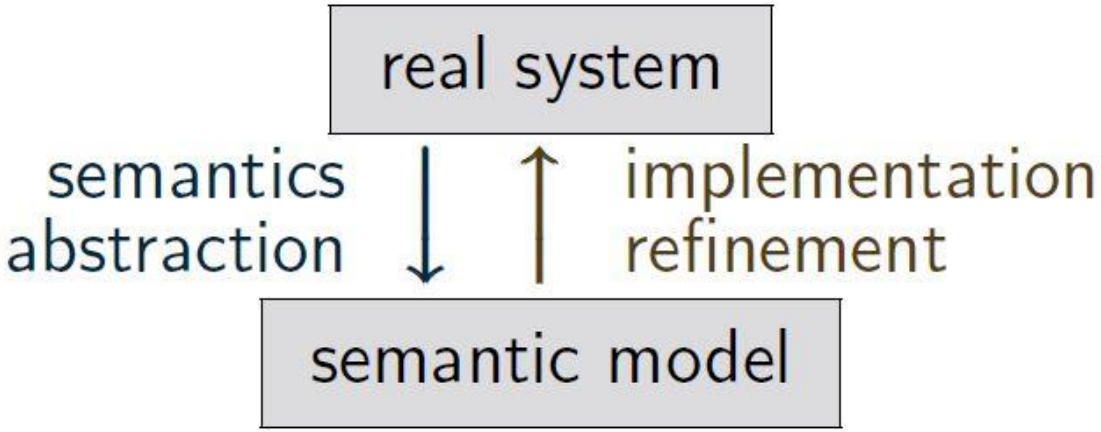
Transition systems



Transition systems



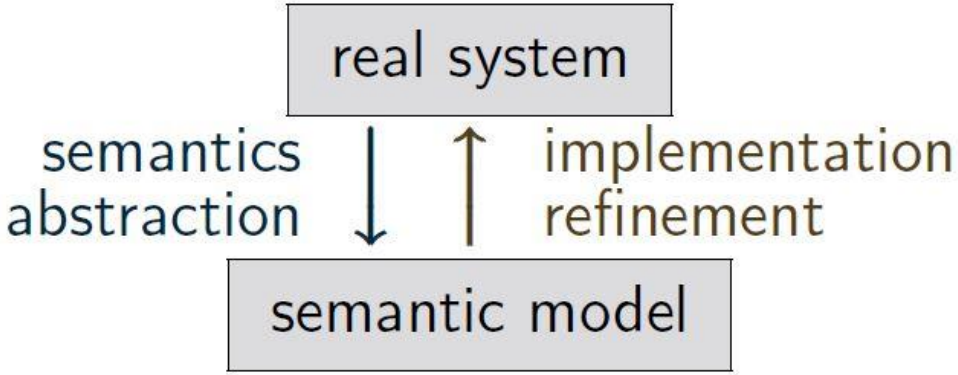
Transition systems



The semantic model yields a formal representation of:

- the states of the system
- the stepwise behaviour
- the initial states

Transition systems



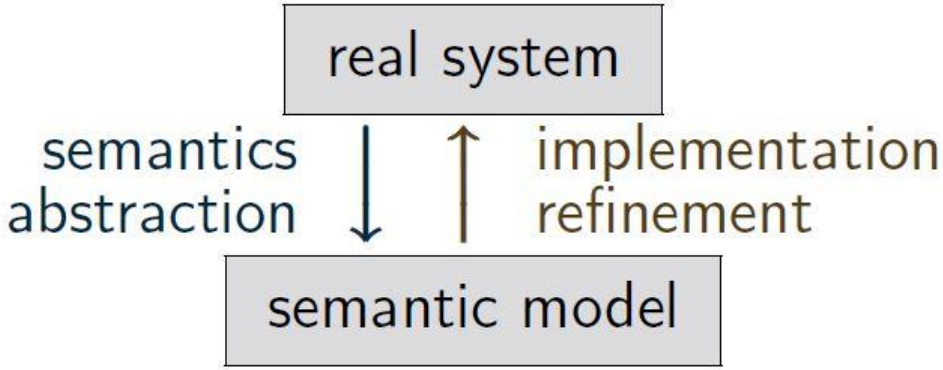
The semantic model yields a formal representation of:

- the states of the system

control component + information on "relevant" data

- the stepwise behaviour
- the initial states

Transition systems ≜ extended graphs



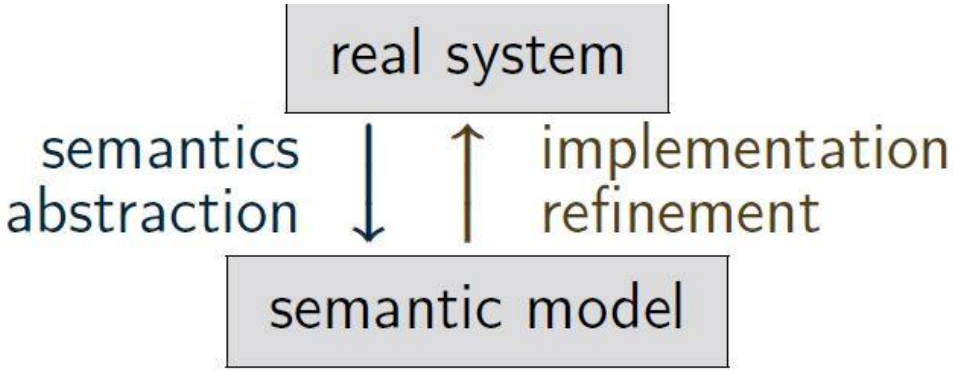
The semantic model yields a formal representation of:

- the states of the system ⟵ **nodes**

control component + information on "relevant" data

- the stepwise behaviour ⟵ **transitions**
- the initial states

Transition systems ≜ extended graphs



The semantic model yields a formal representation of:

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

Atomic propositions (AP)

An *atomic proposition* is a statement or assertion that must be true or false.

Examples of atomic propositions are: "5 is a prime" and "program P terminates".

Propositional formulas are constructed from atomic propositions by using *logical connectives*.

Connectives	
0	false
1	true
$\neg$	not
$\wedge$	and
$\vee$	or
$\rightarrow$	conditional (implies)
$\Leftrightarrow$	biconditional (equivalent)

A typical propositional formula is $A \wedge (B \vee C) \rightarrow B$.

The *truth value* of a propositional formula can be calculated from the truth values of the atomic propositions it contains.

Transition systems (TS)

A transition system is a tuple

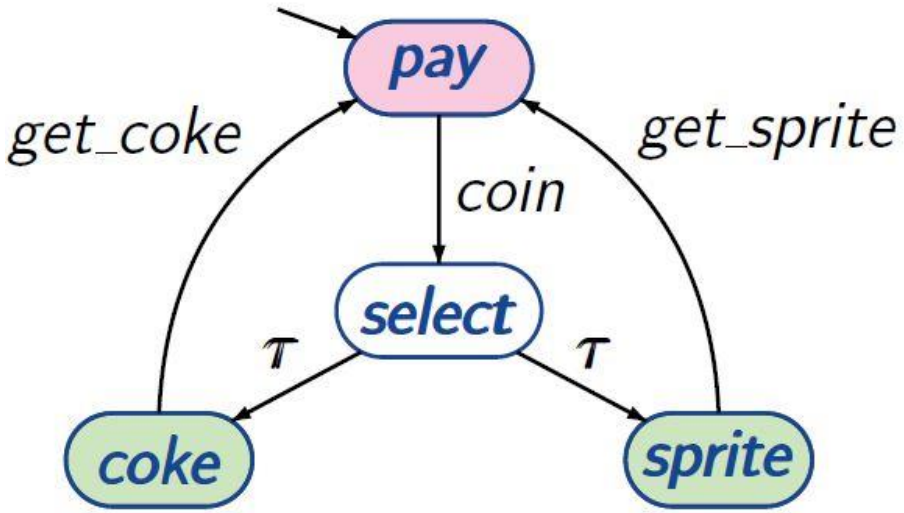
$$\mathcal{T} = (S, Act, \longrightarrow, S_0, AP, L)$$

- 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 systems 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: $L(coke) = L(sprite) = \{drink\}$

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

"Behavior" 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:

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

Possible meanings of nondeterminism in TS (1/8)

- (true) concurrency modeled by interleaving
- competition of parallel dependent actions
- implementational freedom, underspecification
- incomplete information on system environment

TS for parallel actions (2/8)

parallel execution of independent actions ← interleaving

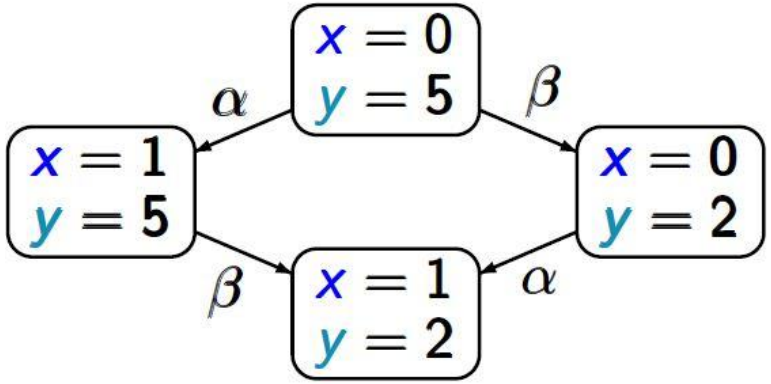
e.g. $\underbrace{x := x+1}_{\text{action } \alpha} \;|||\; \underbrace{y := y-3}_{\text{action } \beta}$ $\quad \alpha, \beta$ independent

parallel execution of dependent actions ← competition

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

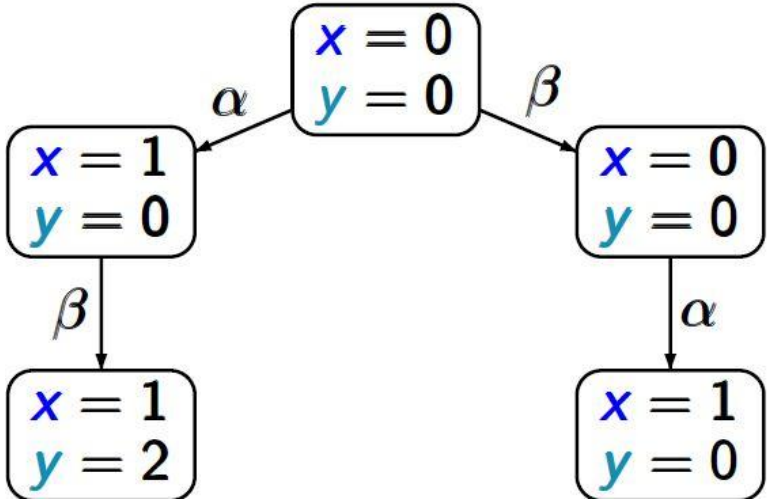
TS for parallel actions (3/8)

parallel execution of independent actions ← interleaving



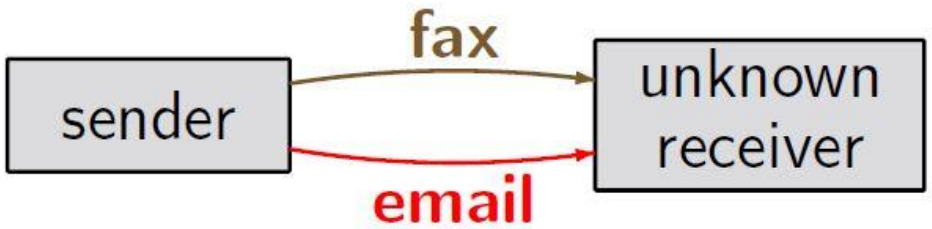
$\underbrace{x := x+1}_{\text{action } \alpha} \;|||\; \underbrace{y := y-3}_{\text{action } \beta}$

parallel execution of dependent actions ← competition

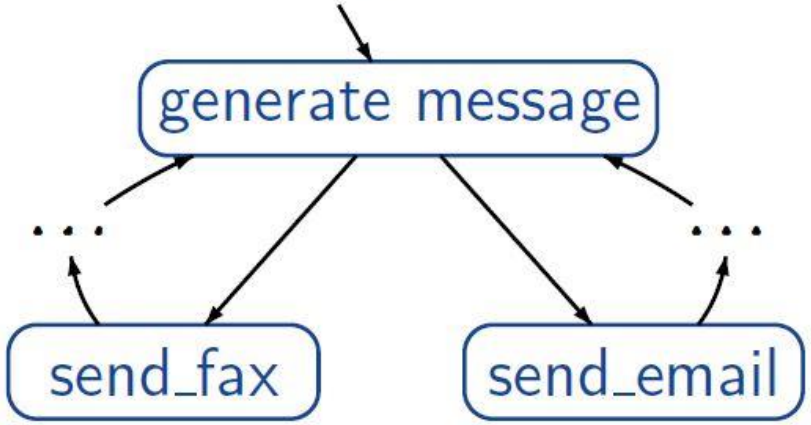


$\underbrace{x := x+1}_{\text{action } \alpha} \;|||\; \underbrace{y := 2*x}_{\text{action } \beta}$

Implementation freedom (4/8): …modelled by nondeterminism

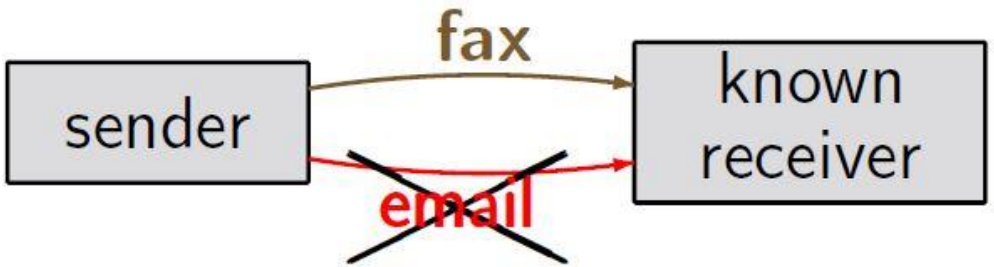


realization by a TS:



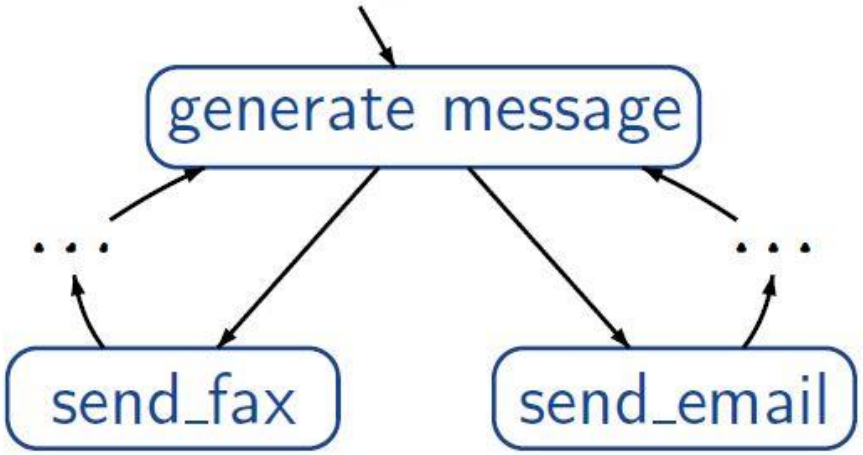
at a future refinement step the nondeterminism is replaced with one of the alternatives

Implementation freedom (5/8)

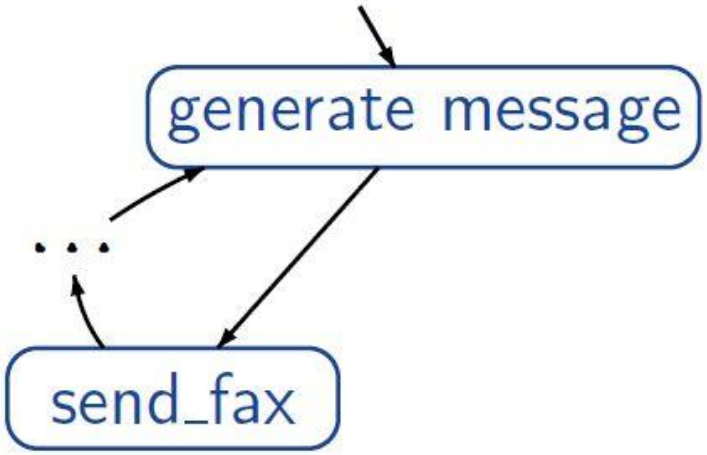


without email access

realization by a TS:

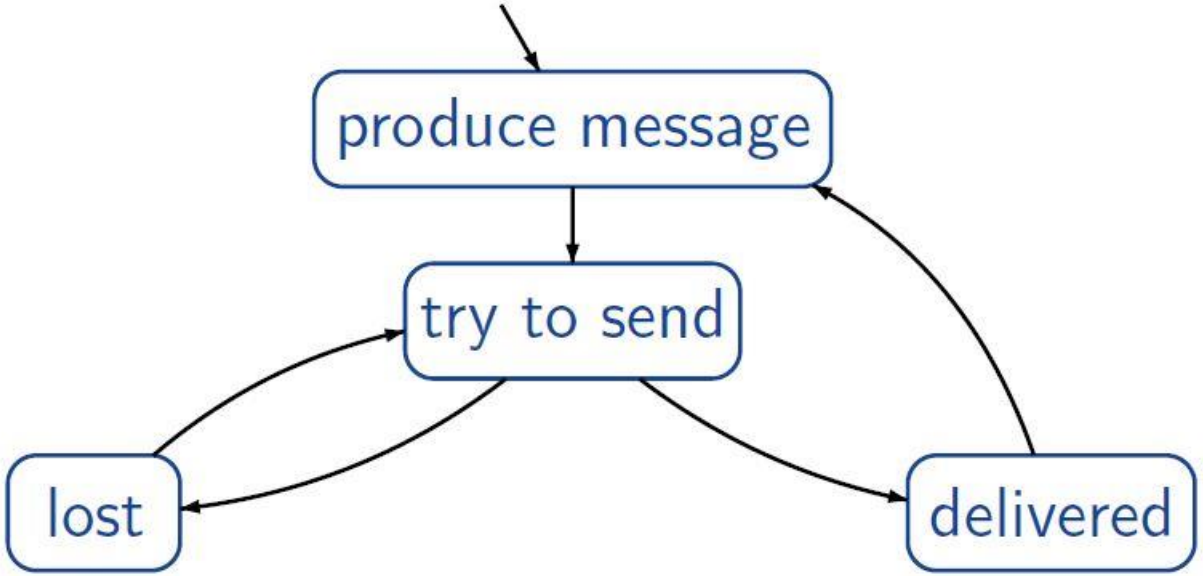


refined TS:

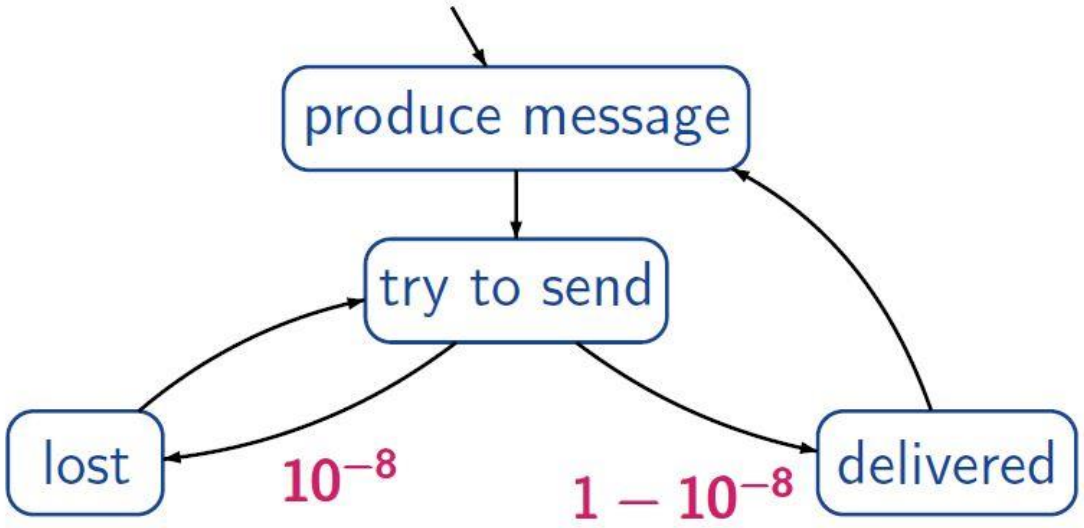


at a future refinement step the nondeterminism is replaced with one of the alternatives

Underspecification (6/8)

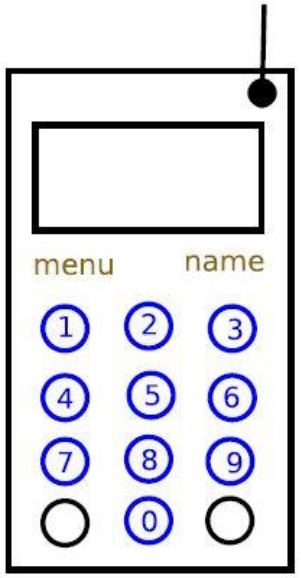


Underspecification (7/8)

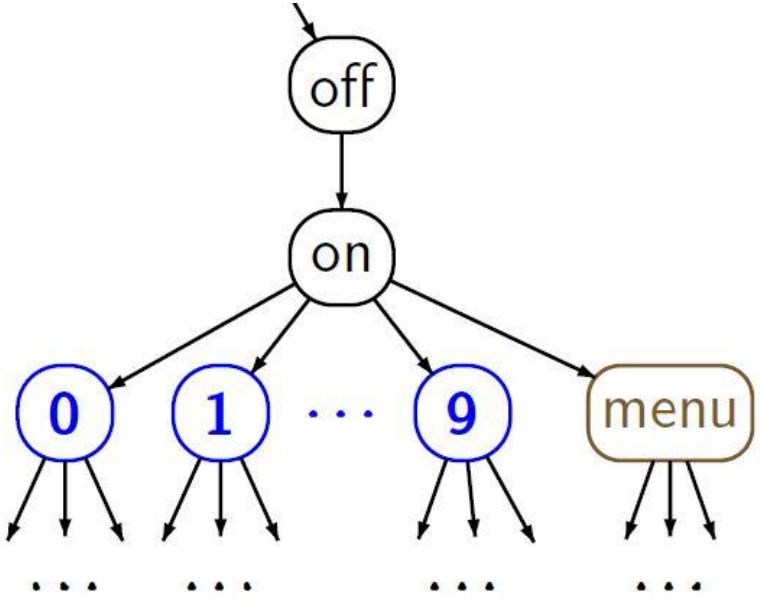


at a future refinement step the nondeterminism is replaced with probabilism

Incomplete information on the environment (8/8)



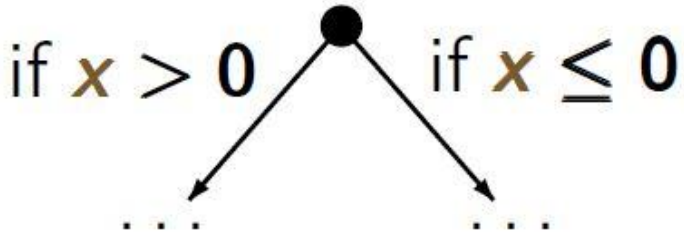
mobile phone



resolution of the nondeterministic choices
by a human user

Data-dependent systems

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

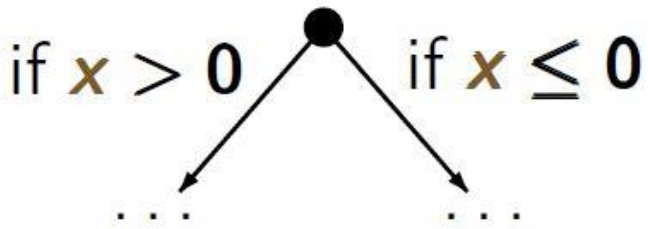


example: sequential program

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

Data-dependent systems

problem: TS-representation of conditional branchings ?

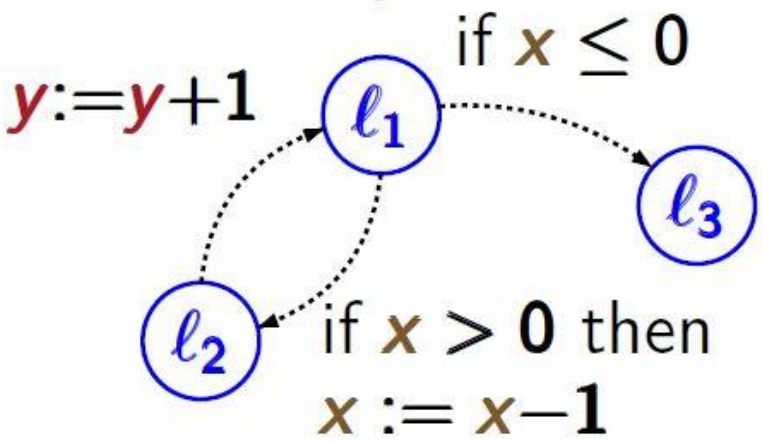


example: sequential program

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

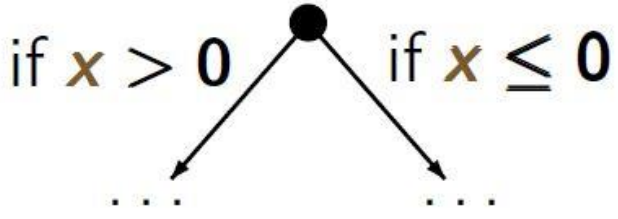
ℓ_1, ℓ_2, ℓ_3 are locations,
i.e., control states

program graph



Data-dependent systems

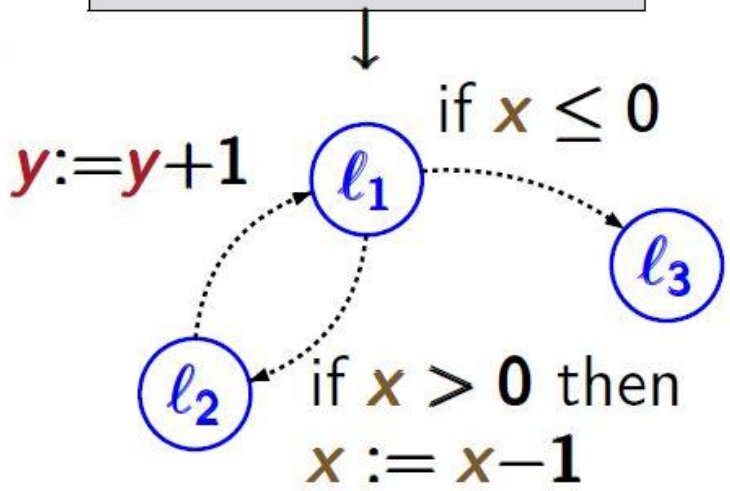
problem: TS-representation of conditional branchings **?**



example: sequential program

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

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

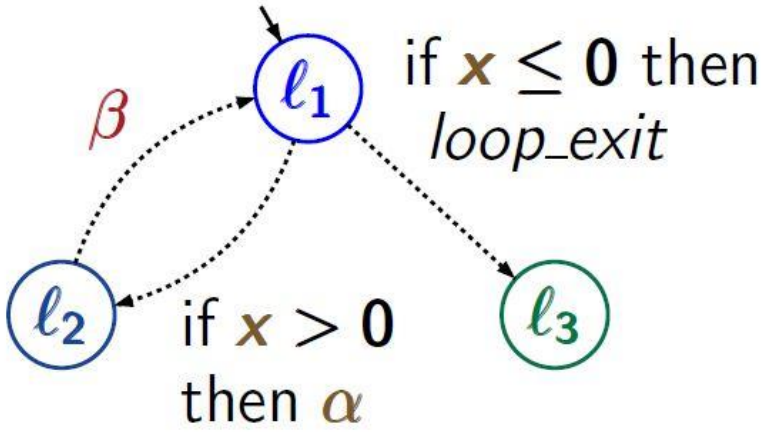
$\quad x := x-1 \leftarrow$ action α

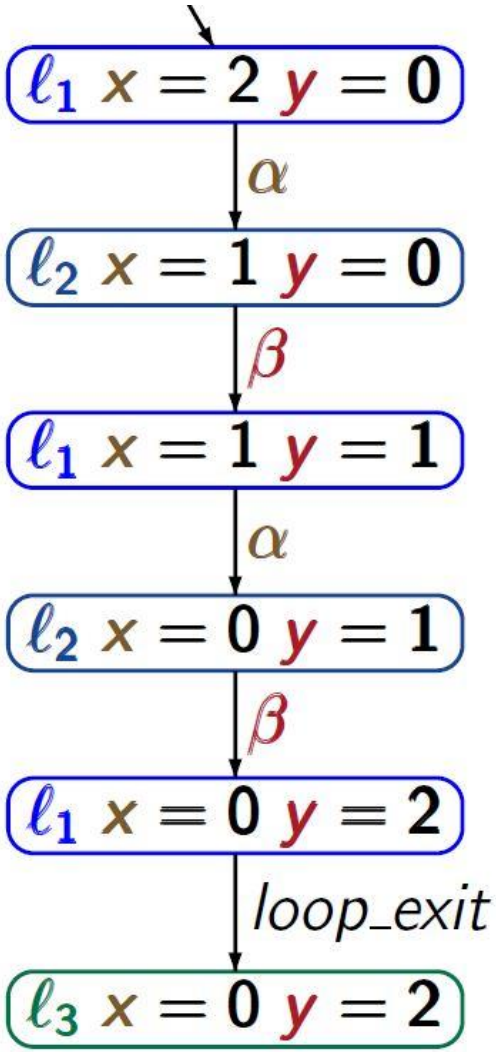
$\ell_2 \rightarrow \quad y := y+1 \leftarrow$ action β

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 $Dom(z) = \{\textbf{yellow}, \textbf{red}, \textbf{blue}\}$

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

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:

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

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 $\ell \xhookrightarrow{g:\alpha} \ell'$

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

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

TS-semantics of a PG

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 PG

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:

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

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

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

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

TS-semantics of a PG

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:

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

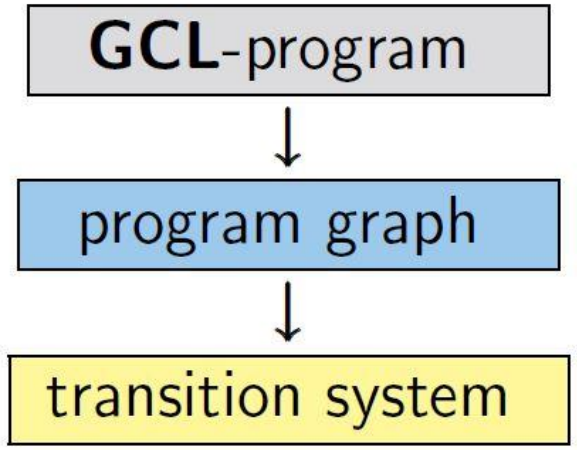
- atomic propositions: $AP = Loc \cup Cond(Var)$
- labeling function:

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

Guarded Command Language (GCL)

by Dijkstra

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



Guarded Command Language (GCL)

guarded command $g \Rightarrow$ ***stmt*** $\leftarrow$ enabled if g is true

repetitive command/loop:

DO :: $g \Rightarrow$ ***stmt*** OD $\leftarrow$ WHILE g DO ***stmt*** OD

conditional command:

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

$\leftarrow$

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

$$
\begin{array}{lcl}
\mathit{stmt} & \stackrel{\text{def}}{=} & x := \mathit{expr} \;\mid\; \mathit{stmt}_1; \mathit{stmt}_2 \;\mid \\
& & \texttt{DO} \;::g_1 \Rightarrow \mathit{stmt}_1 \;\; \ldots \;\; ::g_n \Rightarrow \mathit{stmt}_n \;\texttt{OD} \\
& & \texttt{IF} \;::g_1 \Rightarrow \mathit{stmt}_1 \;\; \ldots \;\; ::g_n \Rightarrow \mathit{stmt}_n \;\texttt{FI} \\
& & \vdots
\end{array}
$$

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

uses the following actions:

	enabled	effect
get_coke	if $\#coke > 0$	$\#coke := \#coke - 1$
get_sprite	if $\#sprite > 0$	$\#sprite := \#sprite - 1$
refill	any time	$\#sprite := max$ $\#coke := max$
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; (* 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

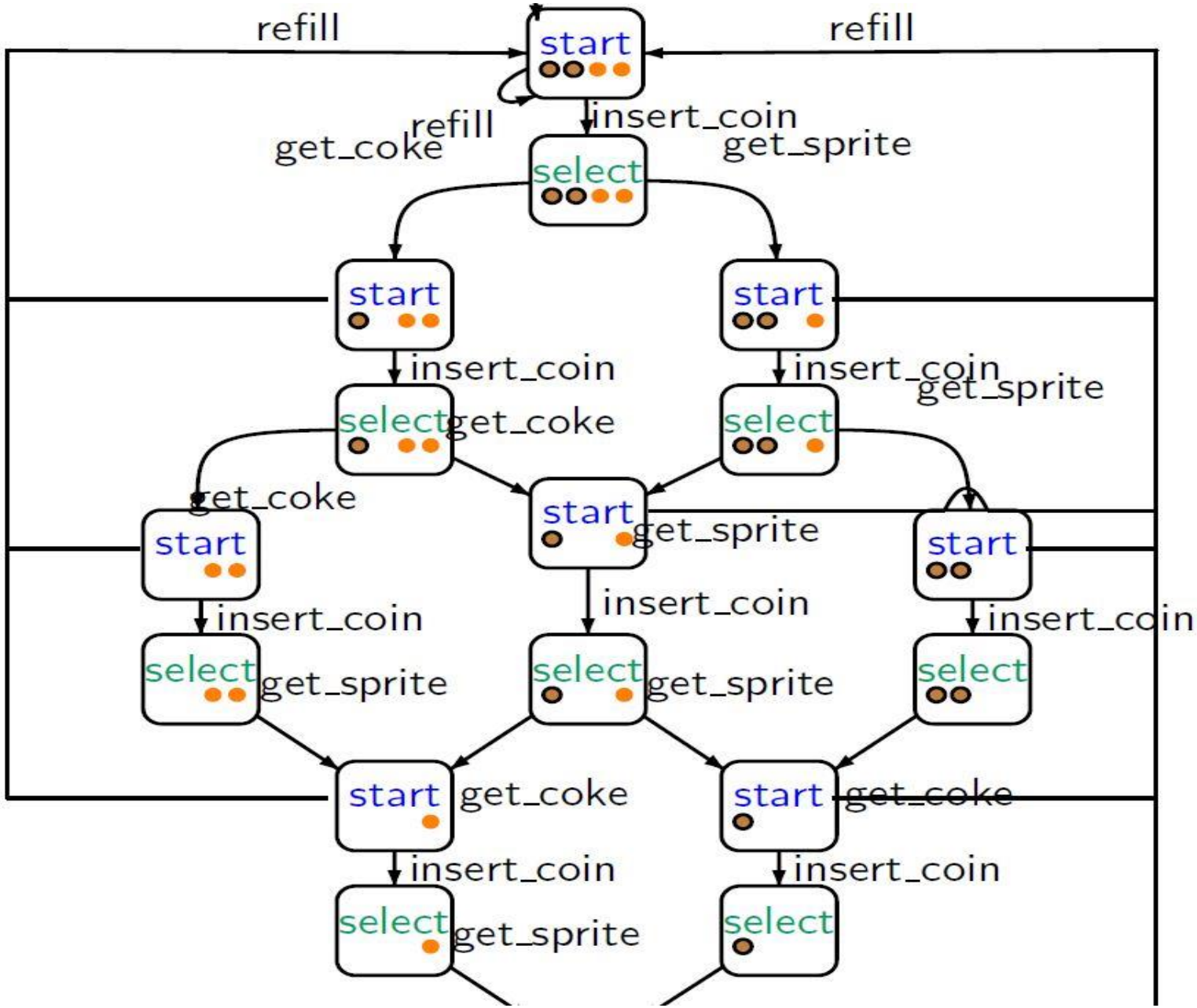
```
DO  ::  true ⇒ insert_coin; (* user inserts a coin *)
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
                                (* no beverage available *)
            ::  #coke  > 0 ⇒ get_coke
                                (* user selects coke *)
            ::  #sprite > 0 ⇒ get_sprite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

GCL-program for beverage machine

$$
\begin{array}{lll}
\textit{start} \rightarrow \texttt{DO} & :: & \text{true} \Rightarrow \text{insert_coin}; \\
\textit{select} \rightarrow & & \texttt{IF} \;\; :: \;\; \#\textit{sprite} = \#\textit{coke} = 0 \\
 & & \qquad\qquad\qquad\qquad \Rightarrow \text{return_coin} \\
 & & \quad\;\; :: \;\; \#\textit{coke} > 0 \Rightarrow \text{get_coke} \\
 & & \quad\;\; :: \;\; \#\textit{sprite} > 0 \Rightarrow \text{get_sprite} \\
 & & \texttt{FI} \\
 & :: & \text{true} \Rightarrow \text{refill} \\
\texttt{OD} & &
\end{array}
$$

... yields a program graph with

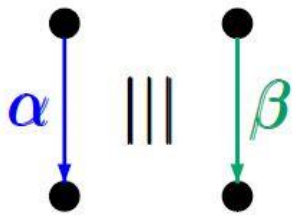
- two variables $\#\textit{sprite}, \#\textit{coke} \in \{0, 1, \ldots, \textit{max}\}$
- two locations *start* and *select*

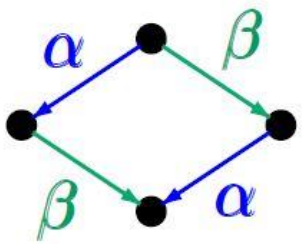

refill
start
refill
refill
insert_coin
get_coke
get_sprite
select
start
start
insert_coin
insert_coin
get_sprite
select
get_coke
select
get_coke
start
get_sprite
start
insert_coin
insert_coin
insert_coin
select
get_sprite
select
get_sprite
select
start
get_coke
start
get_coke
insert_coin
insert_coin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

Interleaving operator ||| *for TS*

- interleaving of concurrent, independent actions of parallel processes (modelled by TS)
- representation by nondeterministic choice:
  "which subprocess performs the next step?"

$$\mathit{effect}(\alpha|||\beta) \;=\; \mathit{effect}(\alpha;\beta+\beta;\alpha)$$





parallel execution of α and β on *two processors* $\;\widehat{=}\;$ serial execution on a *single processor* in arbitrary order

Interleaving operator ||| for TS

$$\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, S_{0,1}, AP_1, L_1)$$

$$\mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, S_{0,2}, AP_2, L_2)$$

The transition system $\mathcal{T}_1 \,|||\, \mathcal{T}_2$ is defined by:

$$\mathcal{T}_1 \,|||\, \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, S_{0,1} \times S_{0,2}, AP, L)$$

where the transition relation $\longrightarrow$ is given by:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

atomic propositions: $AP = AP_1 \uplus AP_2$

labeling function: $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$

- https://github.com/kangeu/Fall2015

- Lab course will start in 2 – 3 weeks

- Find your team members by Oct 22nd :

  - Each group consists of three people.
  - Elect a team leader.
  - Each team leader provides details of his/her group (English Names/Student IDs of members) to Prof. Kang by Oct 22nd.
  - In case you don't provide your group info by the deadline, you will be assigned in a random group by the instructor.