



Modelling Software Systems

Readings:
Peled ch. 4 (section 13 not covered)

Different System Views

- Functional/relational/sequential
 - System as input/output relation, or function
 - Or as datatype with (sequential) operations defined on it
- Structural
 - To account for component structure and interconnections
 - E.g. OO-based modelling
- Reactive
 - Ongoing behaviour important
 - To model concurrency/distribution
 - Or interactions with an unknown environment
- Criteria
 - What system properties are important?
 - Functional correctness? Termination? Deadlocks? Livelocks?

Concurrent/Interactive Systems

Events + nondeterminism during computation significant

Basic model: State transition system

- State = (global) state of system according to modelling abstraction
- Transition = atomic computation step

Possible execution models:

- Linear:
 - System = set of possible linear computation paths
- Branching:
 - System = set of possible computation trees

This course:

- Both models. Linear for now, branching later

The Fundamental Model

Labelled transition system: $T = \langle Q, \Sigma, R, Q_0 \rangle$ where

- Q : Set (finite or infinite) of states q
- Σ : Set (finite or infinite) of actions α, β
- $R \subseteq Q \times \Sigma \times Q$: The transition relation
- $Q_0 \subseteq Q$: Set of Initial states (when needed)

Actions:

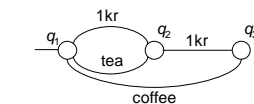
- Anything convenient to model the computation step
- Could be observable events across a process interface
- Could be assignments to state variables, e.g. *Busy*: $Q \rightarrow \{0,1\}$

Transitions:

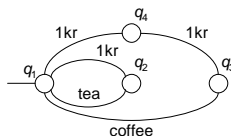
- $q \xrightarrow{\alpha} q'$ means $(q, \alpha, q') \in R$
- Transitions are atomic
- Non-determinism is allowed

Example 1: Coffee Machine

$Q = \{q_1, q_2, q_3\}$
 $Q_0 = \{q_1\}$
 $\Sigma = \{1kr, \text{tea}, \text{coffee}\}$
 $\rightarrow =$
 $\{(q_1, 1kr, q_2), (q_2, \text{tea}, q_1),$
 $(q_2, 1kr, q_3), (q_3, \text{coffee}, q_1)\}$



Compare with:



Are these two machine interchangeable?

State Spaces as in Peled

X : Set of variables x_0, \dots, x_n over domain $dom(x_i)$
 v_i : Values in $dom(x_i)$

State q : assignment $x_i \mapsto v_i \in dom(x_i)$

Example:

$q = \langle x_1 = 3, x_2 = 55, x_3 = \text{"foo"} \rangle$

$\Sigma = \{ \cdot \}$ (we forget Σ for remainder of lesson)

$Q_0 =$ some set of initial states

$\rightarrow =$ some relation on $Q \times Q$

$\rightarrow \cdot$: set of atomic multi-assignments

Example 2

```
int x=0, y=0 ;
cobegin x := x+1 mod 3 | y := y+1 mod 4 coend
```

$$Q = \{ \langle x=i, y=j \rangle \mid 0 \leq i \leq \text{maxint}, 0 \leq j \leq \text{maxint} \}$$

$$Q_0 = \{ \langle x=0, y=0 \rangle \}$$

$\rightarrow =$

$$\{ (q_1, q_2) \mid q_2(x) = q_1(x), q_2(y) = q_1(y) + 1 \text{ mod } 4 \} \cup \{ (q_1, q_2) \mid q_2(x) = q_1(x) + 1 \text{ mod } 3, q_2(y) = q_1(y) \}$$

Transition System Specification

Transition specification:
 $\phi \rightarrow (x_1, \dots, x_n) := (e_1, \dots, e_n)$

where

- ϕ : enabledness condition, first-order predicate
- $q \models \phi$: ϕ is true in state q
- e_1, \dots, e_n : first-order side effect-free expressions
- $e_i(q)$: value of e_i in state q

Example:

$$i > j \rightarrow (i, j) := (j, i)$$

Initial condition ϕ_0

Transition system specification: First-order domain specification + set of transition specifications + initial condition

Example 3

$$X = \{a, b, c, d, e\}$$

$$\text{dom}(x) = \mathbb{N}, \text{ the natural numbers}$$

Transition specs:

1. $c > 0 \rightarrow (c, e) := (c-1, e+1)$
2. $d > 0 \rightarrow (d, e) := (d-1, e+1)$

Initial condition:

$$\phi_0: c=a \wedge d=b \wedge e=0$$

What does the system do?

Generated State Space

Assume symbolic transition system specification

Generated state space is (Q, R, Q_0) where

- Q = Set of states (for given first-order structure)
- $R = \{ (q, q') \mid \text{exists transition spec } \phi \rightarrow (x_1, \dots, x_n) := (e_1, \dots, e_n) \text{ such that} \}$
 - $q \models \phi$
 - $q' = q[x_i \mapsto e_i(q), \dots, x_n \mapsto e_n(q)]$
- $Q_0 = \{ q \mid q \models \phi_0 \}$

Transitions must be enabled in the current state

Resulting state obtained by evaluating all e_i in current state, then performing multiple assignment

Sometimes add also self-loops $q \rightarrow q$ when no transition from q is enabled by a transition spec

Example 4

For the symbolic transition system in ex 3:

$$q_0 = \langle a=2, b=1, c=2, d=1, e=0 \rangle \rightarrow$$

(satisfies initial condition)

$$q_1 = \langle a=2, b=1, c=1, d=1, e=1 \rangle \rightarrow$$

(first transition fires)

$$q_2 = \langle a=2, b=1, c=1, d=0, e=2 \rangle \rightarrow$$

(second transition fires)

$$q_3 = \langle a=2, b=1, c=0, d=0, e=3 \rangle (\rightarrow q_3)$$

(first transition fires)

Exercise: For the generated state space of ex 3 prove that it has the *diamond property*, i.e. that whenever $q \rightarrow q_1$ and $q \rightarrow q_2$ and $q_1 \neq q_2$ then there is a q_3 such that $q_1 \rightarrow q_3$ and $q_2 \rightarrow q_3$. Why is this called the diamond property?

Reachable States

Not all states are necessarily reachable from an initial state

Example: In ex 3 the state $\langle a=2, b=3, c=4, d=2, e=1 \rangle$ is not reachable from any initial state.

Exercise: Prove this statement.

Two possibilities:

1. Ignore the problem and include as valid all states, reachable or not
 Advantage: State space easily defined, many problem expressible as reachability problems
2. Restrict attention to reachable states
 Advantage: State space easily pictured, smaller, more intuitive

For now adopt policy 1, in later lectures revert to 2

Example 5

Computing the binomial coefficient (Manna-Pnueli, Peled):

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$$

$$= \frac{n \times (n-1) \times \dots \times (n-k+1)}{1 \times 2 \times \dots \times k}$$

Two processes

Left: Multiply numerator

Right: Divide denominator

Initial values:

$y_1 = n$ (numerator),
 $y_2 = 0$ (denominator),
 $y_3 = 1$ (result)

```

while  $y_1 \neq n - k$  do      while  $y_2 \neq k$ 
   $y_3 := y_3 \times y_1;$        $y_2 := y_2 + 1;$ 
   $y_1 := y_1 - 1$           await  $y_2 \leq n - y_1;$ 
od                           $y_3 := y_3 / y_2$ 
                             od
    
```

Representing the Program

Using labels and gotos:

```

 $l_1$ : if  $y_1 = n - k$  then halt       $r_1$ : if  $y_2 = k$  then halt
 $l_2$ :  $y_3 := y_3 \times y_1$            ||  $r_2$ :  $y_2 := y_2 + 1$ 
 $l_3$ :  $y_1 := y_1 - 1$                ||  $r_3$ : await  $y_2 \leq n - y_1$ 
 $l_4$ : goto  $l_1$                       $r_4$ :  $y_3 := y_3 / y_2$ 
                                      $r_5$ : goto  $r_1$ 
    
```

Labels: $L_l = \{l_1, \dots, l_4, \text{halt}\}$, $L_r = \{r_1, \dots, r_5, \text{halt}\}$

States: $\langle y_1 \in \mathbb{N}, y_2 \in \mathbb{N}, y_3 \in \mathbb{N}, pc_l \in L_l, pc_r \in L_r \rangle$

Initial condition:

$$\phi_0: y_1 = n \wedge y_2 = 0 \wedge y_3 = 1 \wedge pc_l = l_1 \wedge pc_r = r_1$$

Transition Specifications

```

 $pc_l = l_1 \rightarrow pc_l := \text{if } y_1 = n - k \text{ then halt, else } l_2$ 
 $pc_l = l_2 \rightarrow (pc_l, y_3) := (l_3, y_3 \times y_1)$ 
 $pc_l = l_3 \rightarrow (pc_l, y_1) := (l_4, y_1 - 1)$ 
 $pc_l = l_4 \rightarrow pc_l := l_1$ 
 $pc_r = r_1 \rightarrow pc_r := \text{if } y_2 = k \text{ then halt, else } r_2$ 
 $pc_r = r_2 \rightarrow (pc_r, y_2) := (r_3, y_2 + 1)$ 
 $pc_r = r_3 \wedge y_2 \leq n - y_1 \rightarrow pc_r := r_4$ 
 $pc_r = r_4 \rightarrow (pc_r, y_3) := (r_5, y_3 / y_2)$ 
 $pc_r = r_5 \rightarrow pc_r := r_1$ 
 $pc_l = \text{halt}_l \wedge pc_r = \text{halt}_r \rightarrow () := ()$ 
    
```

Obs: Assignments must be atomic!

Nondeterministic Transitions

Each transition specification determines a unique successor state

Suppose we wish to represent user input:

$$pc = l_n \rightarrow (pc, x) := (l_{n+1}, \text{"some } n\text{"})$$

Not doable in current set-up

Introduce new variables x' , y' etc

Primed variables x' , y' represent value of x , y after the transition has been taken

Express transition specification as first-order formula, e.g.

$$pc = l_n \wedge pc' = l_{n+1} \wedge y' = f(x, y)$$

No constraints on x' : x' can be chosen freely

System spec = initial condition + disjunction of transition specs

Scheduling and Fairness

Initial condition: $x=0$ and $y=0$

```

P1:      P2: while  $y = 0$  do
 $x := 1$  || (noop [] if  $x = 1$  then  $y := 1$ )
           od
    
```

[] is nondeterministic, uncontrollable choice

As transition system specification:

```

T_1:  $pc_l = l_0 \rightarrow (pc_l, x) := (\text{halt}_l, 1)$ 
T_2:  $pc_r = r_0 \wedge y \neq 0 \rightarrow pc_r := \text{halt}_r$ 
T_3:  $pc_r = r_0 \wedge y = 0 \rightarrow pc_r := r_1$ 
T_4:  $pc_r = r_1 \rightarrow pc_r := r_0$ 
T_5:  $pc_r = r_1 \wedge x = 1 \rightarrow (pc_r, y) := (r_0, 1)$ 
T_6:  $pc_l = \text{halt}_l \wedge pc_r = \text{halt}_r \rightarrow () := ()$ 
    
```

Does the system halt?

Fairness Conditions

Weak process fairness:

- For all processes P_i , no execution can from some state onward forever have a transition of P_i enabled, but no transition of P_i is ever taken

Strong process fairness:

- For all processes P_i , no execution can infinitely often have a transition of P_i enabled, but transitions from P_i are executed only a finite number of times

Weak transition fairness:

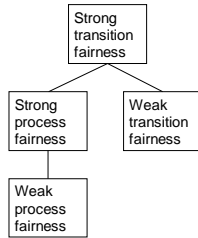
- For all transitions T_i , no execution can from some state onward forever have T_i enabled, but never taken

Strong transition fairness:

- For all transitions T_i , no execution can infinitely often have T_i enabled, but T_i is taken only finite number of times

PS: Many other fairness conditions exists

Hierarchy and Realizability



Scheduler: Algorithm for determining from current state and history which transition to take

"Proposition": No scheduler can exist which generates exactly the strongly transition fair execution sequences

Practical schedulers enforce a fairness constraint by allowing only subset of permitted executions