Program correctness

Transition systems

Marcello Bonsangue
Formal Verification

Verification techniques comprise:

- a modelling framework $M, \Gamma$
  to describe a system

- a specification language $\phi$
  to describe the properties to be verified

- a verification method $M \models \phi, \Gamma \vdash \phi$
  to establish whether a model satisfies a property
Transition Systems

- A very general modelling framework
- Intuitively: a system evolves from one state to another under the action of a transition

A modulo 3 counter
Example: an assignment

States: $s$: Var $\rightarrow$ Val

where $s' = s[s(x) + 1/x]$ and

$$f[v/x](y) = \begin{cases} f(y) & \text{if } x \neq y \\ v & \text{if } x = y \end{cases}$$
Example: a digicode

- 3 keys: A, B, C
- The door open when ABA is keyed
Digicode’s executions

- 1
- 11, 12
- 111, 112, 121, 122, 123
- ...

![Diagram of Digicode's executions]
A few definitions

- **Transition system**: $\langle S, L, \to \rangle$
  - $S$: set of states
  - $L$: set of transition labels
  - $\to \subseteq S \times L \times S$: transition relation

- **Path**: a sequence $\pi$ of infinite transitions which follow each other

For example

$3 \stackrel{B}{\to} 1 \stackrel{A}{\to} 2 \stackrel{A}{\to} 2 \ldots$

is a path of the digicode
Adding data

- Real-life systems consist of control and data. We can model them by
  - control = states+transitions
  - data = state variables

- A transition system interact with state variables in two ways
  - guards: a transition cannot occur if the condition does not hold
  - assignment: a transition can modify the value of some state variables
We do not tolerate more than 3 mistakes (recorded by the variable m)
Unfolding

From a theoretical point of view, transition systems with state variables are not strictly necessary, as we can **unfold** them into ordinary transition systems.

- The new states correspond to the old ones + a component for each variable giving its value
- no more guards and assignment on the new transitions
Unfolding: example

The digicode with error counting

Slide 11
Composing systems

- Systems often consists of cooperating subsystems. Next we describe how to obtain a global transition system form its subsystem by having them cooperate.

- There are many ways to cooperate:
  - product (no interaction)
  - synchronous product
    - by message passing
    - by asynchronous channels
    - by shared variables
Product

- Subsystems do not interact with each other
- The resulting transition system $<S,L,\rightarrow>$ is the cartesian product of the transition systems $<S_1,L_1,\rightarrow>, \ldots, <S_n,L_n,\rightarrow>$ representing the subsystems

  - $S = S_1 \times \ldots \times S_n$
  - $L = L_1 \times \ldots \times L_n$
  - $<s_1,\ldots,s_n> \rightarrow <t_1,\ldots,t_n> \text{ if for all } i, s_i \rightarrow t_i$
Example

- Few transitions of the product of two modulo 3 counters

```
(0,0) -> (1,0) inc,inc
(1,0) -> (0,1) inc,inc
(0,1) -> (1,1) inc,inc
(1,1) -> (1,2) inc,dec
(1,2) -> (2,2) inc,inc
(0,2) -> (1,2) inc,inc
(2,2) -> (0,2) inc,inc
```

Synchronized Product

- Subsystems interact by doing some step together (synchronization).

- To synchronize subsystems we restrict the transitions allowed in their cartesian product.

- A synchronization set
  \[ \text{Sync} \subseteq L_1 \times \ldots \times L_n \]
  define the labels of those transitions corresponding to a synchronization. Transitions with other labels are forbidden.
Example

- Few transitions of two counters counting at the same time

\[ \text{Sync} = \{ <\text{inc},\text{inc}>, <\text{dec},\text{dec}> \} \]
**Example**

- Few transitions of two counters counting one at the time

$$\text{Sync} = \{ <\text{inc},->, <\text{dec},->, <-,\text{inc}>, <-,\text{dec}> \}$$
Message Passing

- A special case of synchronized product
- Two special sets of labels
  - $!m$ emission of message $m$
  - $?m$ reception of message $m$

- In message passing, only transitions in which a given emission is executed simultaneously with the corresponding reception will be permitted
Example: An elevator

- An elevator in a three floors building consists of
  - a cabin which goes up and down
  - three doors which open and close
  - a controller which commands the three doors and the cabin

- Elevator requests from people at one of the three floors are not modeled, as they are the environment outside the system
Example: An elevator

- The cabin

- The i-th door
Example: An elevator

The controller

- free2
  - !close(2)
  - !open(2)
- on2
  - !up
  - !down
- 2->0
  - !up

- free1
  - !close(1)
  - !open(1)
- on1
  - !up
- 0->2
  - !up
  - !down

- free0
  - !close(0)
  - !open(0)
- on0
  - !up
  - !down
Example: An elevator

- The synchronization

- Sync =

{<\?open(0),-, -, -, !open(0)>, <\?close(0), -, -, -, !close(0)>,
<-, ?open(1), -, -, !open(1)>, <-, ?close(1), -, -, !close(1)>,
<-, -, ?open(2), -, !open(2)>, <-, -, ?close(2), -, !close(2)>,
<-, -, -, ?down, !down>, <-, -, -, ?up, !up>}

Leiden Institute of Advanced Computer Science
Asynchronous Messages

- Like message passing, but messages are not received instantly.
- Emitted messages but not yet received remain in a communication channel, usually a FIFO buffer.
- A communication channel can be modeled by a transition system with a variable (for the buffer content).
Example:

- **Producer**
  
  $x=0 \xrightarrow{!send(x)} x:=x+1$

- **Buffer**
  
  $buf=\varepsilon \xrightarrow{?send(x)} buf:=buf\oplus x$
  
  $buf=y \rightarrow ?receive(y)$
  
  $buf=y \oplus w \rightarrow ?receive(y); buf:=w$

- **Consumer**
  
  $y=0 \xrightarrow{?receive(y)}$