Concurrency
Lecture #3 of Model Checking

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling and Verification

E-mail: katoen@cs.rwth-aachen.de

April 10, 2007
Overview Lecture #3

⇒ Concurrency

- The interleaving paradigm

- Communication principles
  - Shared variable “communication”
  - Handshaking
  - Synchronous communication

- Channel systems

- The state-space explosion problem
Concurrent systems

- Transition systems
  - suited for modeling sequential data-dependent systems
  - and for modeling sequential hardware circuits

- How about *concurrent* systems?
  - threading
  - distributed algorithms and communication protocols

- Can we model:
  - threading without communication?
  - synchronous communication?
  - synchronous composition of hardware?
Interleaving

- Abstract from decomposition of system in components

- Actions of independent components are merged or “interleaved”
  - a single processor is available
  - on which the actions of the processes are interlocked

- No assumptions are made on the order of processes
  - possible orders for non-terminating independent processes $P$ and $Q$:
    
    $\begin{align*}
    P & \quad Q & \quad P & \quad Q & \quad P & \quad Q & \quad P & \quad \ldots \\
    P & \quad P & \quad Q & \quad P & \quad P & \quad Q & \quad P & \quad \ldots \\
    P & \quad Q & \quad P & \quad P & \quad Q & \quad P & \quad P & \quad Q & \quad \ldots \\
    \end{align*}$

  - assumption: there is a scheduler with an a priori unknown strategy
Interleaving

- Justification for interleaving:
  
  the effect of concurrently executed, independent actions $\alpha$ and $\beta$ equals
  the effect when $\alpha$ and $\beta$ are successively executed in arbitrary order

- Symbolically this is stated as:

  $\text{Effect}(\alpha ||| \beta, \eta) = \text{Effect}((\alpha ; \beta) + (\beta ; \alpha), \eta)$

  - $|||$ stands for the (binary) interleaving operator
  - ";" stands for sequential execution, and "+" for non-deterministic choice
Interleaving

\[ x := x + 1 \quad \mid\mid\mid \quad y := y - 2 \]

\[
\begin{align*}
x &:= x + 1 \\
\Rightarrow &\quad = \alpha \\
y &:= y - 2 \\
\Rightarrow &\quad = \beta
\end{align*}
\]

\[
\begin{array}{ccc}
x=0 & \mid\mid\mid & y=7 \\
\downarrow \alpha & & \downarrow \beta \\
x=1 & & y=5
\end{array}
\]

\[
\begin{array}{ccc}
x=0, y=7 & \mid\mid\mid & x=1, y=7 \\
\downarrow \alpha & & \downarrow \beta \\
x=0, y=5 & & x=1, y=5
\end{array}
\]

\[
\begin{array}{ccc}
x=0, y=7 & \mid\mid\mid & x=1, y=7 \\
\downarrow \beta & & \downarrow \alpha \\
x=0, y=5 & & x=1, y=5
\end{array}
\]
Interleaving of transition systems

Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, $i=1, 2$, be two transition systems

Transition system

$$TS_1 ||| TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\rightarrow$ is defined by the rules:

$$s_1 \xrightarrow{\alpha_1} s_1'$$
$$\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle$$

and

$$s_2 \xrightarrow{\alpha_2} s_2'$$
$$\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle$$
What are program graphs?

A program graph $PG$ over set $Var$ of typed variables is a tuple

$$(Loc, Act, Effect, \rightarrow, Loc_0, g_0)$$

where

- $Loc$ is a set of locations with initial locations $Loc_0 \subseteq Loc$
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ is the effect function
- $\rightarrow \subseteq Loc \times (\underset{\text{Boolean conditions over } Var}{\underbrace{\text{Cond}(Var)}} \times Act) \times Loc$, transition relation
- $g_0 \in \text{Cond}(Var)$ is the initial condition.

Notation: $\ell \xrightarrow{g, \alpha} \ell'$ denotes $(\ell, g, \alpha, \ell') \in \rightarrow$
Beverage vending machine

- $\text{Loc} = \{ \text{start, select} \}$ with $\text{Loc}_0 = \{ \text{start} \}$

- $\text{Act} = \{ \text{bget, sget, coin, ret\_coin, refill} \}$

- $\text{Var} = \{ \text{nsprite, nbeer} \}$ with domain $\{ 0, 1, \ldots, \text{max} \}$

\[
\begin{align*}
\text{Effect}(\text{coin}, \eta) &= \eta \\
\text{Effect}(\text{ret\_coin}, \eta) &= \eta \\
\text{Effect}(\text{sget}, \eta) &= \eta[\text{nsprite} := \text{nsprite} - 1] \\
\text{Effect}(\text{bget}, \eta) &= \eta[\text{nbeer} := \text{nbeer} - 1] \\
\text{Effect}(\text{refill}, \eta) &= [\text{nsprite} := \text{max}, \text{nbeer} := \text{max}] \\
\end{align*}
\]

- $g_0 = (\text{nsprite} = \text{max} \land \text{nbeer} = \text{max})$
From program graphs to transition systems

• Basic strategy: *unfolding*
  
  – state = location (current control) $\ell$ + data valuation $\eta$
  – initial state = initial location satisfying the initial condition $g_0$

• Propositions and labeling
  
  – propositions: “at $\ell$” and “$x \in D$” for $D \subseteq \text{dom}(x)$
  – $\langle \ell, \eta \rangle$ is labeled with “at $\ell$” and all conditions that hold in $\eta$

• $\ell \xrightarrow{g;\alpha} \ell'$ and $g$ holds in $\eta$ then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$
Transition systems for program graphs

The transition system $TS(PG)$ of program graph

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

over set $\text{Var}$ of variables is the tuple $(S, \text{Act}, \rightarrow, I, \text{AP}, L)$ where

- $S = \text{Loc} \times \text{Eval}(\text{Var})$
- $\rightarrow \subseteq S \times \text{Act} \times S$ is defined by the rule: 
  $$\frac{\ell \xrightarrow{\text{g:}\alpha} \ell' \land \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle}$$
- $I = \{ \langle \ell, \eta \rangle | \ell \in \text{Loc}_0, \eta \models g_0 \}$
- $\text{AP} = \text{Loc} \cup \text{Cond}(\text{Var})$ and $L(\langle \ell, \eta \rangle) = \{ \ell \} \cup \{ g \in \text{Cond}(\text{Var}) | \eta \models g \}$. 
Interleaving of program graphs

For program graphs $PG_1$ (on $Var_1$) and $PG_2$ (on $Var_2$) without shared variables, i.e., $Var_1 \cap Var_2 = \emptyset$,

$$TS(PG_1) ||| TS(PG_2)$$

faithfully describes the concurrent behavior of $PG_1$ and $PG_2$

what if they have variables in common?
Shared variable communication

\[
\begin{align*}
&\text{action } \alpha : \quad x := 2 \cdot x \\
&\text{action } \beta : \quad x := x + 1
\end{align*}
\]

with initially \( x = 3 \)

\[
\begin{array}{ccc}
& x=3 & \\
\alpha & \quad & \beta \\
& x=6 & \quad & x=4 & \\
\end{array}
\]

\[
\begin{array}{ccc}
& x=6, x=3 & \\
\alpha & \quad & \beta \\
& x=3, x=3 & \quad & x=3, x=4 & \\
\end{array}
\]

\[
\langle x=6, x=4 \rangle \text{ is an inconsistent state!}
\]

\[\Rightarrow\] no faithful model of the concurrent execution of \( \alpha \) and \( \beta \)

Idea: first unfold, then interleave
Interleaving of program graphs

Let $PG_i = (Loc_i, Act_i, Effect_i, \rightarrow_i, Loc_{0,i}, g_{0,i})$ over variables $Var_i$.

Program graph $PG_1 ||| PG_2$ over $Var_1 \cup Var_2$ is defined by:

$$(Loc_1 \times Loc_2, Act_1 \cup Act_2, Effect, \rightarrow, Loc_{0,1} \times Loc_{0,2}, g_{0,1} \land g_{0,2})$$

where $\rightarrow$ is defined by the inference rules:

$$
\ell_1 \xrightarrow{g:\alpha_1} \ell'_1 \quad \text{and} \quad \ell_2 \xrightarrow{g:\alpha_2} \ell'_2
$$

and $Effect(\alpha, \eta) = Effect_i(\alpha, \eta)$ if $\alpha \in Act_i$. 

© JPK
Example

\[ x := 2 \cdot x \quad ||| \quad x := x + 1 \]

action \( \alpha \)
action \( \beta \)

with initially \( x = 3 \)

note that \( TS(PG_1) ||| TS(PG_2) \neq TS(PG_1 ||| PG_2) \)
On atomicity

\[
x := x + 1; y := 2x + 1; z := y \text{ div } x
\]
\[
\text{non-atomic}
\]

Possible execution fragment:

\[
\langle x = 11 \rangle \xrightarrow{x:=x+1} \langle x = 12 \rangle \xrightarrow{y:=2x+1} \langle x = 12 \rangle \xrightarrow{x:=0} \langle x = 0 \rangle \xrightarrow{z:=y/x} \ldots
\]

\[
\langle x := x + 1; y := 2x + 1; z := y \text{ div } x \rangle \text{ ||| } x := 0
\]

atomic

Either the left process or the right process is completed first:

\[
\langle x = 11 \rangle \xrightarrow{x:=x+1} \langle x = 12 \rangle \xrightarrow{y:=2x+1} \langle x = 12 \rangle \xrightarrow{z:=y/x} \langle x = 12 \rangle \xrightarrow{x:=0} \langle x = 0 \rangle
\]
Peterson’s mutual exclusion algorithm

\[ P_1 \quad \text{loop forever} \]
\[ : \quad \text{(* non-critical actions *)} \]
\[ \langle b_1 := \text{true}; x := 2 \rangle ; \quad \text{(* request *)} \]
\[ \text{wait until } (x = 1 \lor \neg b_2) \]
\[ \text{do critical section od} \]
\[ b_1 := \text{false} \quad \text{(* release *)} \]
\[ : \quad \text{(* non-critical actions *)} \]
\[ \text{end loop} \]

\( b_i \) is true if and only if process \( P_i \) is waiting or in critical section
if both processes want to enter their critical section, \( x \) decides who gets access
Banking system

Person Left behaves as follows:

```plaintext
while true {
    ......
    nc : ⟨b₁, x = true, 2;⟩
    wt : wait until(x == 1 || ¬ b₂) {
    cs : . . . @account . . .
    b₁ = false;
    ......
    }
}
```

Person Right behaves as follows:

```plaintext
while true {
    ......
    nc : ⟨b₂, x = true, 1;⟩
    wt : wait until(x == 2 || ¬ b₁) {
    cs : . . . @account . . .
    b₂ = false;
    ......
    }
}
```

Can we guarantee that only one person at a time has access to the bank account?
Program graph representation
Is the banking system safe?

Manually inspect whether two may have access to the account simultaneously: No
Banking system with non-atomic assignment

Person Left behaves as follows:

```plaintext
while true {
    ......
    nc : x = 2;
    rq : b_1 = true;
    wt : wait until(x == 1 | \neg b_2) {
        cs : ...@account ...
        b_1 = false;
        ......
    }
}
```

Person Right behaves as follows:

```plaintext
while true {
    ......
    nc : x = 1;
    rq : b_2 = true;
    wt : wait until(x == 2 | \neg b_1) {
        cs : ...@account ...
        b_2 = false;
        ......
    }
}
```
On atomicity again

Assume that the location inbetween the assignments $x := \ldots$ and $b_i := \text{true}$ in program graph $PG_i$ is called $rq_i$. Possible state sequence:

\[
\langle nc_1, \quad nc_2, \quad x = 1, \quad b_1 = \text{false}, \quad b_2 = \text{false} \rangle
\]
\[
\langle nc_1, \quad rq_2, \quad x = 1, \quad b_1 = \text{false}, \quad b_2 = \text{false} \rangle
\]
\[
\langle rq_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{false}, \quad b_2 = \text{false} \rangle
\]
\[
\langle wt_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{false} \rangle
\]
\[
\langle cs_1, \quad rq_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{false} \rangle
\]
\[
\langle cs_1, \quad wt_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{true} \rangle
\]
\[
\langle cs_1, \quad cs_2, \quad x = 2, \quad b_1 = \text{true}, \quad b_2 = \text{true} \rangle
\]

violation of the mutual exclusion property
Parallelism and handshaking

- Concurrent processes run truly in parallel
- To obtain cooperation, some interaction mechanism is needed
- If processes are distributed there is no shared memory

⇒ Message passing
  - synchronous message passing (= handshaking)
  - asynchronous message passing (= channel communication)
Handshaking

- Concurrent processes interact by \textit{synchronous message passing}
  - processes execute synchronized actions together
  - that is, in interaction both processes need to participate at the same time
  - the interacting processes “shake hands”

- Abstract from information that is exchanged

- $H$ is a set of \textit{handshake actions}
  - actions outside $H$ are independent and are interleaved
  - actions in $H$ need to be synchronized
Handshaking

Let $TS_i = (S_i, Act_i, \to_i, I_i, AP_i, L_i)$, $i=1, 2$ and $H \subseteq Act_1 \cap Act_2$

$$TS_1 \parallel_H TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \to, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where $L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$ and with $\to$ defined by:

- $s_1 \xrightarrow{\alpha_1} s'_1 \quad \langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s_2 \rangle$ for $\alpha \notin H$

- $s_2 \xrightarrow{\alpha_2} s'_2 \quad \langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s'_2 \rangle$ for $\alpha \notin H$

- $s_1 \xrightarrow{\alpha_1} s'_1 \land s_2 \xrightarrow{\alpha_2} s'_2 \quad \langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s'_2 \rangle$ for $\alpha \in H$

Note that $TS_1 \parallel_H TS_2 = TS_1 \parallel_H TS_2$ but $(TS_1 \parallel_H TS_2) \parallel_H TS_3 \neq TS_1 \parallel_H (TS_2 \parallel_H TS_3)$
A booking system

\[ BCR \parallel BP \parallel Printer \]

\( \parallel \) is a shorthand for \( \parallel_H \) with \( H = \text{Act}_1 \cap \text{Act}_2 \)
The parallel composition
Pairwise handshaking

$TS_1\parallel\ldots\parallel TS_n$ for $H_{i,j} = \text{Act}_i \cap \text{Act}_j$ with $H_{i,j} \cap \text{Act}_k = \emptyset$ for $k \notin \{i, j\}$

State space of $TS_1\parallel\ldots\parallel TS_n$ is the Cartesian product of those of $TS_i$

- for $\alpha \in \text{Act}_i \setminus \left( \bigcup_{0<j\leq n \atop i \neq j} H_{i,j} \right)$ and $0 < i \leq n$:

\[
\begin{align*}
    s_i \xrightarrow{\alpha} i s'_i \\
    \langle s_1, \ldots, s_i, \ldots, s_n \rangle \xrightarrow{\alpha} \langle s_1, \ldots, s'_i, \ldots, s_n \rangle
\end{align*}
\]

- for $\alpha \in H_{i,j}$ and $0 < i < j \leq n$:

\[
\begin{align*}
    s_i \xrightarrow{\alpha} i s'_i \quad \land \quad s_j \xrightarrow{\alpha} j s'_j \\
    \langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n \rangle \xrightarrow{\alpha} \langle s_1, \ldots, s'_i, \ldots, s'_j, \ldots, s_n \rangle
\end{align*}
\]
Synchronous parallelism

Let $TS_i = (S_i, Act, \rightarrow_i, I_i, AP_i, L_i)$ and $Act \times Act \rightarrow Act$, $(\alpha, \beta) \rightarrow \alpha \ast \beta$

$TS_1 \otimes TS_2 = (S_1 \times S_2, Act, \rightarrow, I_1 \times I_2, AP_1 \uplus AP_2, L)$

with $L$ as defined before and $\rightarrow$ is defined by the following rule:

$$
\begin{align*}
&s_1 \xrightarrow{\alpha} s'_1 \quad \text{and} \quad s_2 \xrightarrow{\beta} s'_2 \\
&\langle s_1, s_2 \rangle \xrightarrow{\alpha \ast \beta} \langle s'_1, s'_2 \rangle
\end{align*}
$$

typically used for synchronous hardware circuits, cf. next example
#3: Concurrency

**Model checking**

\[ r_1 \quad \text{NOT} \quad y \]

\[ r_2 \quad \text{OR} \quad y' \]

\[ TS_1 : \]

0

1

\[ TS_2 : \]

00

01

10

11

\[ TS_1 \otimes TS_2 : \]

000

100

010

110

111

011

101

001