Introduction to Model Checking

René Thiemann

Institute of Computer Science
University of Innsbruck

WS 2007/2008

Concurrent systems

- Transition systems
  - suited for modeling sequential data-dependent systems (last lecture)
- What about *concurrent* systems?
  - threading
  - distributed algorithms and communication protocols
- Can we model:
  - synchronous communication?
  - asynchronous communication?

Interleaving

- Construct concurrent system from several (sequential) components
- Actions of independent components are merged or interleaved
  - a single or more processors are available (perhaps on different computers)
  - on which the actions of the processes are interlocked
- No assumptions are made on the order of processes
  - possible orders for independent processes P and Q:

$$\begin{array}{cccccccccc} P & Q & P & Q & P & Q & Q & Q & P & \ldots \\ P & P & Q & P & P & Q & P & P & Q & \ldots \\ P & Q & P & P & Q & P & P & P & Q & \ldots \end{array}$$

Justification for interleaving:

- the effect of concurrently executed, independent actions α and β equals
- the effect when α and β are successively executed in arbitrary order

Channels

Usually, processes exchange data in some way ⇒ channels

- Processes communicate via *channels* ($c \in Chan$)
- Channels are first-in, first-out buffers
- Channels are typed (wrt. their content — $dom(c)$)
- Channels buffer messages (of appropriate type)
- Channel capacity = maximum # messages that can be stored
  - c is a channel with finite capacity $cap(c)$
  - if $cap(c) > 0$, there is some "delay" between sending and receipt
  - if $cap(c) = 0$, then communication via c amounts to handshaking

Channels

- Process P_i = *program graph* PG_i + *communication actions*

  $c!v$ transmit the value v along channel c

  $c?x$ receive a message via channel c and assign it to variable x

- $Comm$ =
  $\{\ c!v,\ c?x\ \mid\ c \in Chan,\ v \in dom(c),\ x \in Var.\ \ dom(x) \supseteq dom(c)\ \}$
- Sending and receiving a message
  - $c!v$ puts the value v at the rear of the buffer c (if c is not full)
  - $c?x$ retrieves the front element of the buffer and assigns it to x (if c is not empty)
  - if $cap(c) = 0$, channel c has *no* buffer
  - if $cap(c) = 0$, sending and receiving can takes place simultaneously this is called synchronous message passing or handshaking
  - if $cap(c) > 0$, sending and receiving can never take place simultaneously this is called *asynchronous message passing*

Example: Traffic Light

Channel systems

A program graph over $(Var, Chan)$ is a tuple

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

where

$$\rightarrow\ \subseteq\ \ Loc \times (Cond(Var) \times Act) \times Loc\ \cup\ \underbrace{Loc \times (Cond(Var) \times Comm) \times Loc}_{\text{communication actions}}$$

A *channel system* CS over $(\bigcup_{0<i\leqslant n} Var_i, Chan)$:

$$CS\ =\ [PG_1 \mid \cdots \mid PG_n]$$

with program graphs PG_i over $(Var_i, Chan)$

Example: Traffic lights

Communication actions

Handshaking

- if $cap(c) = 0$, then process P_i can perform $\ell_i \xrightarrow{g:c!v} \ell_i'$ only
- ... if P_j, say, can perform $\ell_j \xrightarrow{g':c?x} \ell_j'$ and ...
- if both g and g' are satisfied, and
- the effect corresponds to the (atomic) *distributed* assignment $x := v$.

Asynchronous message passing

later ...

Transition system semantics of a channel system

Let $CS = [PG_1 \mid \cdots \mid PG_n]$ be a *channel system* over $(Chan, Var)$ with

$$PG_i = (Loc_i, Act_i, Effect_i, \rightarrow_i, Loc_{0,i}, g_{0,i}), \qquad \text{for } 0 < i \leqslant n$$

$TS(CS)$ is the *transition system* $(S, Act, \rightarrow, I, AP, L)$ where:

- $S = (Loc_1 \times \cdots \times Loc_n) \times Eval(Var) \times Eval(Chan)$
- $Act = \left(\biguplus_{0<i\leqslant n} Act_i\right) \uplus \{\tau\}$
- $\rightarrow$ is defined by the inference rules on the next slides
- $I = \left\{ \langle \ell_1, \ldots, \ell_n, \eta, \xi_0 \rangle \mid \forall i.\ (\ell_i \in Loc_{0,i} \ \&\ \eta \models g_{0,i}) \ \&\ \forall c.\ \xi_0(c) = \varepsilon \right\}$
- $AP = \biguplus_{0<i\leqslant n} Loc_i \uplus Cond(Var)$
- $L(\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle) = \{\ell_1, \ldots, \ell_n\} \cup \{g \in Cond(Var) \mid \eta \models g\}$

Inference rules (I)

- Interleaving for $\alpha \in Act_i$:

$$\frac{\ell_i \xrightarrow{g:\alpha}_i \ell_i' \quad \wedge \quad \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi \rangle}$$

  where $\eta' = Effect(\alpha, \eta)$

- Synchronous message passing over $c \in Chan$, $cap(c) = 0$:

$$\frac{\ell_i \xrightarrow{g:c?x}_i \ell_i' \quad \wedge \quad \ell_j \xrightarrow{g':c!v}_j \ell_j' \quad \wedge \quad (\eta \models g \wedge g') \quad \wedge \quad i \neq j}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n, \eta', \xi \rangle}$$

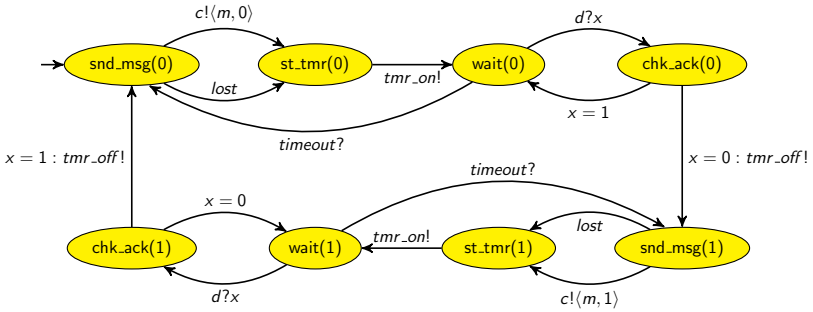
  where $\eta' = \eta[x := v]$.

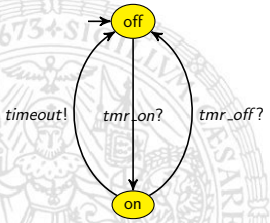
Example: Traffic lights

The alternating bit protocol

- One sender, one receiver
- Two channels
  - a non-reliable, uni-direction channel from sender to receiver
  - a reliable uni-directional channel receiver to sender
- If data is lost, receiver may request resubmission
- Proper modeling requires asynchronous message passing
  - if $cap(c) > 0$, then process P_i can perform $\ell_i \xrightarrow{g:c!v} \ell_i'$
  - ... iff g is satisfied and less than $cap(c)$ messages are stored in c
  - P_j may perform $\ell_j \xrightarrow{g:c?v} \ell_j'$ iff g is satisfied and c is not empty
  - then the first element v of the buffer is extracted and assigned to x (atomically)

	executable if ...	effect
$g : c!v$	g is sat. and c is not full	$Enqueue(c, v)$
$g : c?x$	g is sat. and c is not empty	$x := Front(c)$; $Dequeue(c)$;





Channel evaluations

- A *channel evaluation* ξ is
  - a mapping from channel $c \in Chan$ onto a sequence $\xi(c) \in dom(c)^*$ such that
  - current length cannot exceed the capacity of c: $len(\xi(c)) \leqslant cap(c)$
  - $\xi(c) = v_1\, v_2\, \ldots\, v_k$ ($cap(c) \geqslant k$) denotes v_1 is at front of buffer etc.
- $\xi[c := v_1 \ldots v_k]$ denotes the channel evaluation

$$\xi[c := v_1 \ldots v_k](c') \;=\; \begin{cases} \xi(c') & \text{if } c \neq c' \\ v_1 \ldots v_k & \text{if } c = c'. \end{cases}$$

- Initial channel evaluation ξ_0 equals $\xi_0(c) = \varepsilon$ for any c

Inference rules (II)

- Asynchronous message passing for $c \in Chan$, $cap(c) > 0$:
  - receive a value along channel c and assign it to variable x:

$$\frac{\ell_i \xrightarrow{g:c?x}_i \ell_i' \quad \wedge \quad len(\xi(c)) = k > 0 \quad \wedge \quad \xi(c) = v_1 \ldots v_k \quad \wedge \quad \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta', \xi' \rangle}$$

where $\eta' = \eta[x := v_1]$ and $\xi' = \xi[c := v_2 \ldots v_k]$.

  - transmit value $v \in dom(c)$ over channel c:

$$\frac{\ell_i \xrightarrow{g:c!v}_i \ell_i' \quad \wedge \quad len(\xi(c)) = k < cap(c) \quad \wedge \quad \xi(c) = v_1 \ldots v_k \quad \wedge \quad \eta \models g}{\langle \ell_1, \ldots, \ell_i, \ldots, \ell_n, \eta, \xi \rangle \xrightarrow{\tau} \langle \ell_1, \ldots, \ell_i', \ldots, \ell_n, \eta, \xi' \rangle}$$

where $\xi' = \xi[c := v_1\, v_2 \ldots v_k\, v]$.

Handling unexpected messages

nanoPromela

nanoPromela-program $\overline{\mathcal{P}} = [\mathcal{P}_1|\ldots|\mathcal{P}_n]$ with $\mathcal{P}_i$ processes
A process is specified by a statement:

$$
\begin{array}{lll}
\text{stmt} & ::= & \texttt{skip} \mid x := \text{expr} \mid c?x \mid c!\text{expr} \mid \\
& & \text{stmt}_1 ; \text{stmt}_2 \mid \texttt{atomic}\{\text{assignments}\} \mid \\
& & \textbf{if} \;\; :: g_1 \Rightarrow \text{stmt}_1 \;\; \ldots \;\; :: g_n \Rightarrow \text{stmt}_n \;\; \textbf{fi} \mid \\
& & \textbf{do} \;\; :: g_1 \Rightarrow \text{stmt}_1 \;\; \ldots \;\; :: g_n \Rightarrow \text{stmt}_n \;\; \textbf{od} \\
\text{assignments} & ::= & x_1 := \text{expr}_1 ; x_2 := \text{expr}_2 ; \ldots ; x_m := \text{expr}_m
\end{array}
$$

x is a variable in *Var*, expr an expression and c a channel, g_i a guard

assume the Promela specification is type-consistent

nanoPromela

- Promela (Process Meta Language) is modeling language for SPIN
  - most widely used model checker SPIN
  - developed by Gerard Holzmann (Bell Labs, NASA JPL)
  - ACM Software Award 2002
- nanoPromela is the core of Promela
  - shared variables and channel-based communication
  - formal semantics of a Promela model is a channel system
  - processes are defined by means of a guarded command language
- No actions, statements describe effect of actions

Exercises

- Confirm yourself that it is not a good idea to use asynchronous communication for the traffic lights. To this end, create the transition system for the channel system

  $$[\mathit{TrafficLight} \mid \mathit{TrafficLight} \mid \mathit{Starter}]$$

  where c has a buffer of size 1.