

## 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  $Q$ :

```

P Q P Q P Q Q Q P ...
P P Q P P Q P P Q ...
P Q P P Q P P P Q ...

```

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

## Channels

Usually, processes exchange data in some way  $\Rightarrow$  **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  $P_i = \text{program graph } PG_i + \text{communication actions}$ 
  - $c!v$  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 take 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  $(\bigcup_{0 < i \leq n} Var_i, Chan)$ :

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

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

## Example: Traffic lights

## Communication actions

### Handshaking

- if  $\text{cap}(c) = 0$ , 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  $CS = [PG_1 \mid \dots \mid PG_n]$  be a *channel system* over  $(Chan, Var)$  with

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

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

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

## Inference rules (I)

- Interleaving for  $\alpha \in Act_i$ :

$$\frac{l_i \xrightarrow{g:\alpha} l'_i \quad \wedge \quad \eta \models g}{\langle l_1, \dots, l_i, \dots, l_n, \eta, \xi \rangle \xrightarrow{\alpha} \langle l_1, \dots, l'_i, \dots, l_n, \eta', \xi \rangle}$$

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

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

$$\frac{l_i \xrightarrow{g:c?x} l'_i \quad \wedge \quad l_j \xrightarrow{g':c!v} l'_j \quad \wedge \quad (\eta \models g \wedge g') \quad \wedge \quad i \neq j}{\langle l_1, \dots, l_i, \dots, l_j, \dots, l_n, \eta, \xi \rangle \xrightarrow{\tau} \langle l_1, \dots, l'_i, \dots, l'_j, \dots, l_n, \eta', \xi \rangle}$$

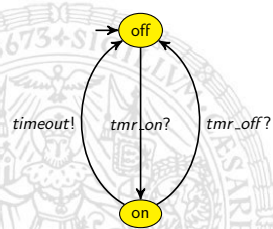
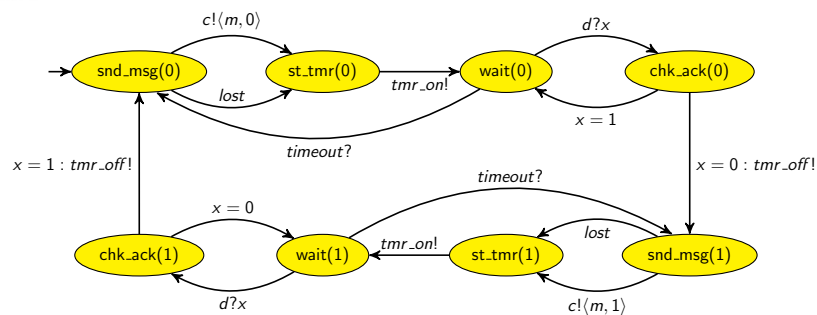
where  $\eta' = \eta[x := v]$ .

## 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	$Enqueue(c, v)$
$g : c?v$	$g$ is sat. and $c$ is not empty	$x := Front(c); Dequeue(c);$



## Channel evaluations

- A *channel evaluation*  $\xi$  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  $c$ :  $len(\xi(c)) \leq cap(c)$
  - $\xi(c) = v_1 v_2 \dots v_k$  ( $cap(c) \geq k$ ) denotes  $v_1$  is at front of buffer etc.
- $\xi[c := v_1 \dots v_k]$  denotes the channel evaluation

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

- Initial channel evaluation  $\xi_0$  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  $x$ :

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

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

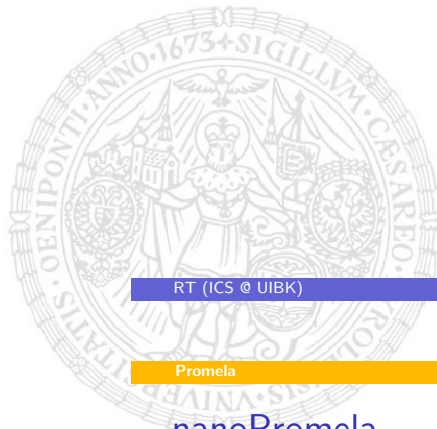
- transmit value  $v \in dom(c)$  over channel  $c$ :

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

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



## Handling unexpected messages



## 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

## nanoPromela

nanoPromela-program  $\bar{\mathcal{P}} = [\mathcal{P}_1 | \dots | \mathcal{P}_n]$  with  $\mathcal{P}_i$  processes

A process is specified by a statement:

```

stmt      ::= skip | x := expr | c?x | c!expr |
           stmt1; stmt2 | atomic{assignments} |
           if :: g1 ⇒ stmt1 ... :: gn ⇒ stmtn fi |
           do :: g1 ⇒ stmt1 ... :: gn ⇒ stmtn od
assignments ::= x1 := expr1; x2 := expr2; ... ; xm := exprm

```

$x$  is a variable in *Var*, *expr* an expression and *c* a channel,  $g_i$  a guard  
assume the Promela specification is type-consistent

## 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

$[TrafficLight | TrafficLight | Starter]$

where *c* has a buffer of size 1.