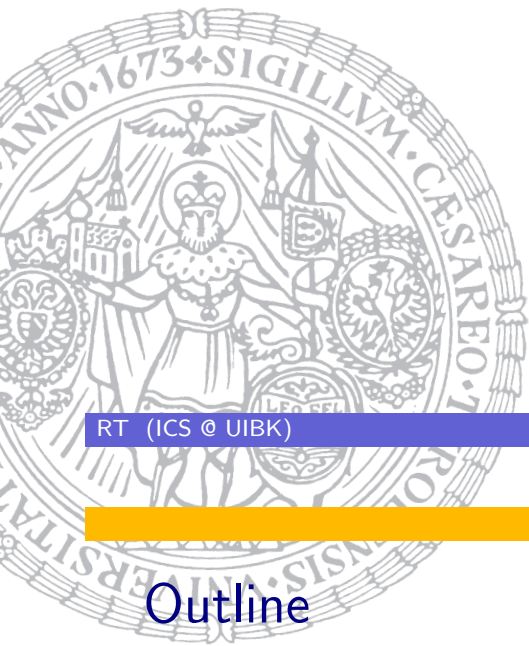


# Introduction to Model Checking

René Thiemann

Institute of Computer Science  
University of Innsbruck

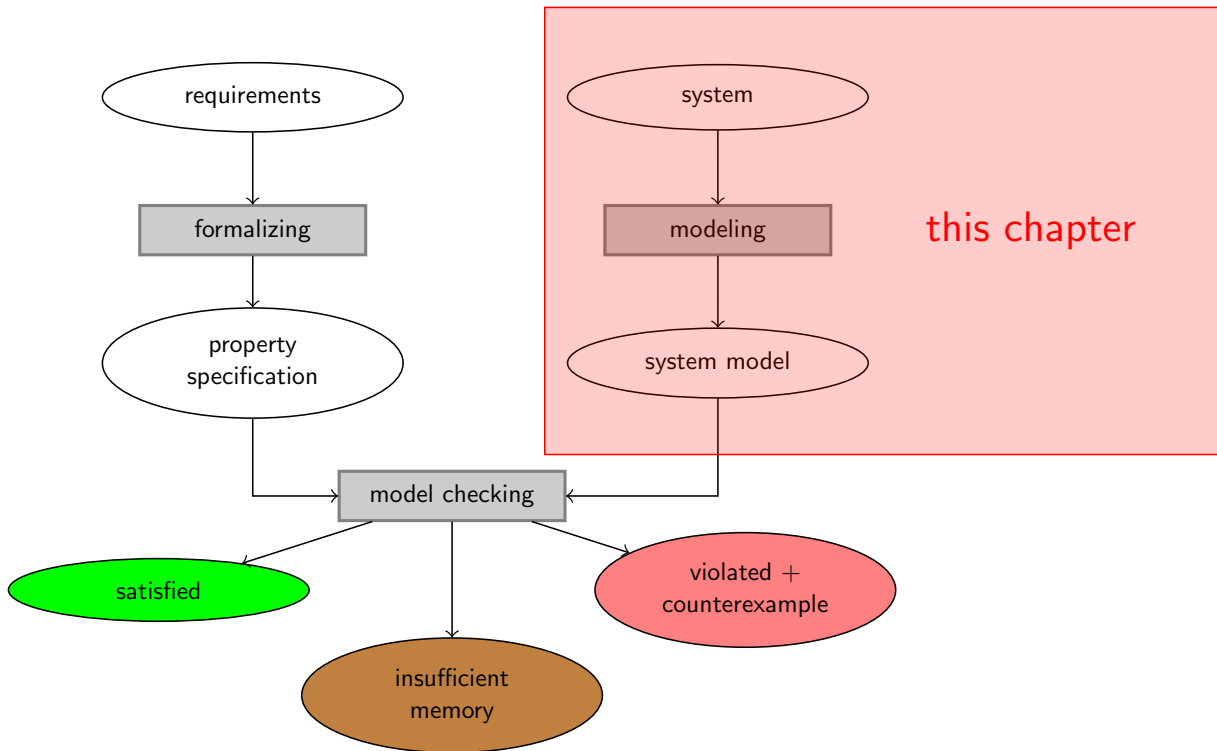
WS 2011/2012



## Outline

- Program Graphs
- Channel systems
- Promela
  - Promela - Syntax and Intuitive Meaning
  - Formal semantics
- The State-Space Explosion Problem

# Model checking overview



## Motivation

- so far, input to model checker is transition system and formula
- for modeling want higher-level description as transition system
  - use variables  
⇒ program graphs
  - use communication  
⇒ channel systems
  - use textual format  
⇒ Promela

# Outline

- Program Graphs
- Channel systems
- Promela
  - Promela - Syntax and Intuitive Meaning
  - Formal semantics
- The State-Space Explosion Problem

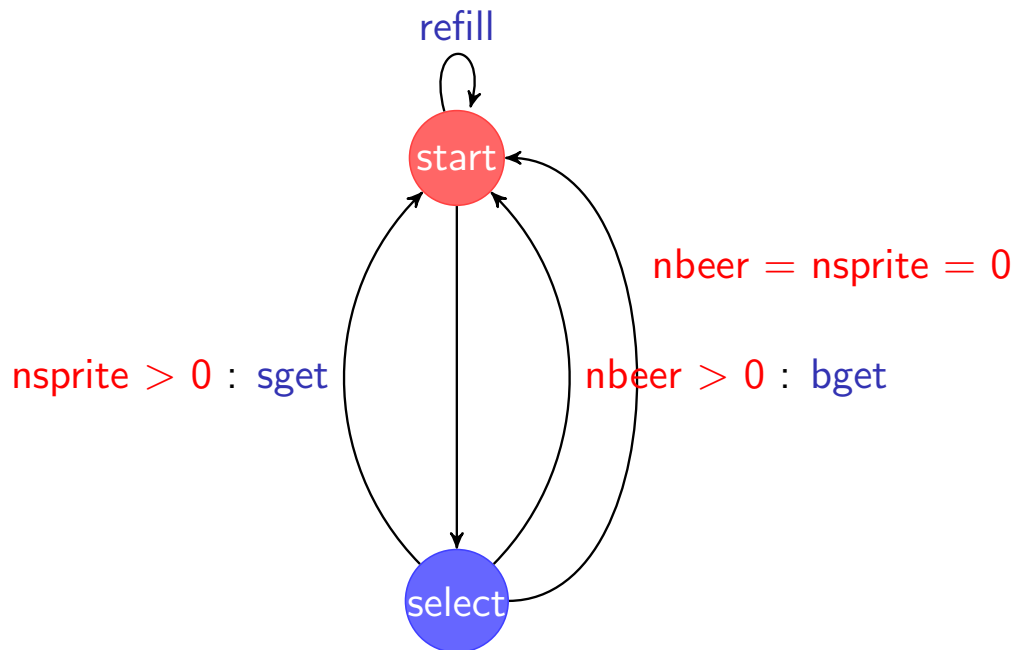
## Beverage vending machine revisited

transitions (with **variables**, **conditions**, and **actions**):

$$\begin{array}{l}
 \text{start} \rightarrow \text{select} \qquad \text{start} \xrightarrow{\text{refill}} \text{start} \\
 \text{select} \xrightarrow{\text{nsprite} > 0 : \text{sget}} \text{start} \qquad \text{select} \xrightarrow{\text{nbeer} > 0 : \text{bget}} \text{start} \\
 \text{select} \xrightarrow{\text{nsprite} = 0 \wedge \text{nbeer} = 0} \text{start}
 \end{array}$$

Action	Effect on variables
<i>sget</i>	$\text{nsprite} := \text{nsprite} - 1$
<i>bget</i>	$\text{nbeer} := \text{nbeer} - 1$
<i>refill</i>	$\text{nsprite} := \text{max}; \text{nbeer} := \text{max}$

# Program graph representation



## Some preliminaries

- typed **variables**  $Var$  with **evaluation**  $\eta$  that assigns values of domain  $D$  to variables
  - e.g.,  $\eta(x) = 17$  and  $\eta(y) = green$
- the set of Boolean **conditions** over  $Var$ 
  - propositional logic formulas whose propositions are of the form " $\bar{x} \in \bar{D}$ "
  - $(nsprite \geq 1) \wedge (y = yellow) \wedge (x \leq 2 \cdot x')$
- **effect** of the actions is formalized by means of a mapping:

$$Effect : Act \times Eval(Var) \rightarrow Eval(Var)$$

- e.g., for action  $\alpha$  use update  $x := y == yellow ? 2 \cdot x : x - 1$ , and evaluation  $\eta$  is given by  $\eta(x) = 17$  and  $\eta(y) = red$
- $Effect(\alpha, \eta)(x) = \eta(x) - 1 = 16$ , and  $Effect(\alpha, \eta)(y) = \eta(y) = red$

## Program graphs

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

$$(Loc, Act, Effect, \longrightarrow, Loc_0, g_0) \quad \text{where}$$

- $Loc$  is a set of **locations** with initial locations  $Loc_0 \subseteq Loc$
  - $Act$  is a set of actions including the empty action  $\sqcup$ , usually not written
  - $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$  is the **effect** function
    - $\sqcup$  has never an effect, i.e.,  $Effect(\sqcup, \eta) = \eta$
  - $\longrightarrow \subseteq Loc \times (Cond(Var) \times Act) \times Loc$ , transition relation
    - $Cond(Var)$ : Boolean conditions over  $Var$  true is not written
  - $g_0 \in Cond(Var)$  is the initial **condition**
- notation:  $l \xrightarrow{g:\alpha} l'$  denotes  $(l, g, \alpha, l') \in \longrightarrow$

## Beverage vending machine

NOT IN HANDOUT, GRAPH IN PRESENTATION

- $Loc = \{ start, select \}$  with  $Loc_0 = \{ start \}$
- $Act = \{ bget, sget, refill \}$
- $Var = \{ nsprite, nbeer \}$  with domain  $\{ 0, 1, \dots, max \}$ 
  - $Effect(sget, \eta) = \eta[nsprite := nsprite - 1]$
  - $Effect(bget, \eta) = \eta[nbeer := nbeer - 1]$
  - $Effect(refill, \eta) = [nsprite := max, nbeer := max]$
- $g_0 = (nsprite = max \wedge nbeer = max)$

# From program graphs to transition systems

- basic strategy: **unfolding**
  - state = location  $\ell$  + evaluation  $\eta$
  - 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  $\eta$
- if  $\ell \xrightarrow{g:\alpha} \ell'$  and  $g$  holds in  $\eta$  then  $\langle \ell, \eta \rangle \rightarrow \langle \ell', \text{Effect}(\alpha, \eta) \rangle$

## Structured operational semantics

- notation  $\frac{\text{premise}}{\text{conclusion}}$  means:
  - if the proposition above the “solid line” (i.e., the premise) holds, then the proposition under the fraction bar (i.e., the conclusion) holds
- such “if ... then ...” propositions are also called **inference rules**
- if the premise is a tautology, it may be omitted
- in the latter case, the rule is also called an **axiom**

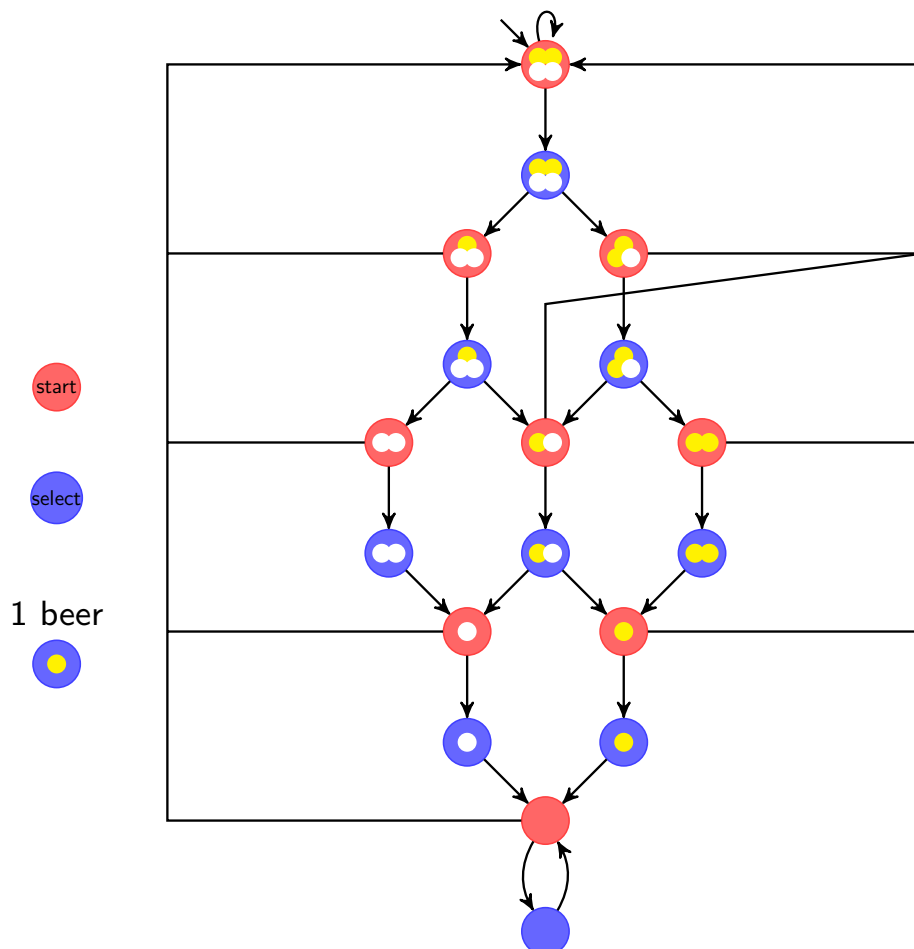
# Transition systems for program graphs

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

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

over set  $Var$  of variables is the tuple  $(S, \longrightarrow, I, AP, L)$  where

- $S = Loc \times Eval(Var)$
- $\longrightarrow \subseteq S \times S$  is defined by the rule: 
$$\frac{l \xrightarrow{g:\alpha} l' \wedge \eta \models g}{\langle l, \eta \rangle \rightarrow \langle l', Effect(\alpha, \eta) \rangle}$$
- $I = \{ \langle l, \eta \rangle \mid l \in Loc_0, \eta \models g_0 \}$
- $AP = Loc \cup Cond(Var)$  and  $L(\langle l, \eta \rangle) = \{l\} \cup \{g \in Cond(Var) \mid \eta \models g\}$



# Outline

- Program Graphs
- Channel systems
- Promela
  - Promela - Syntax and Intuitive Meaning
  - Formal semantics
- The State-Space Explosion Problem

# Concurrent systems

- program graphs
  - suited for modeling **sequential** data-dependent systems
- 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 on the order of processes
  - possible orders for independent processes  $P$  and  $Q$ :

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

- 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
- $P: x++; \dots \quad Q: y++ \dots$ , parallel execution = sequential execution

## Channels

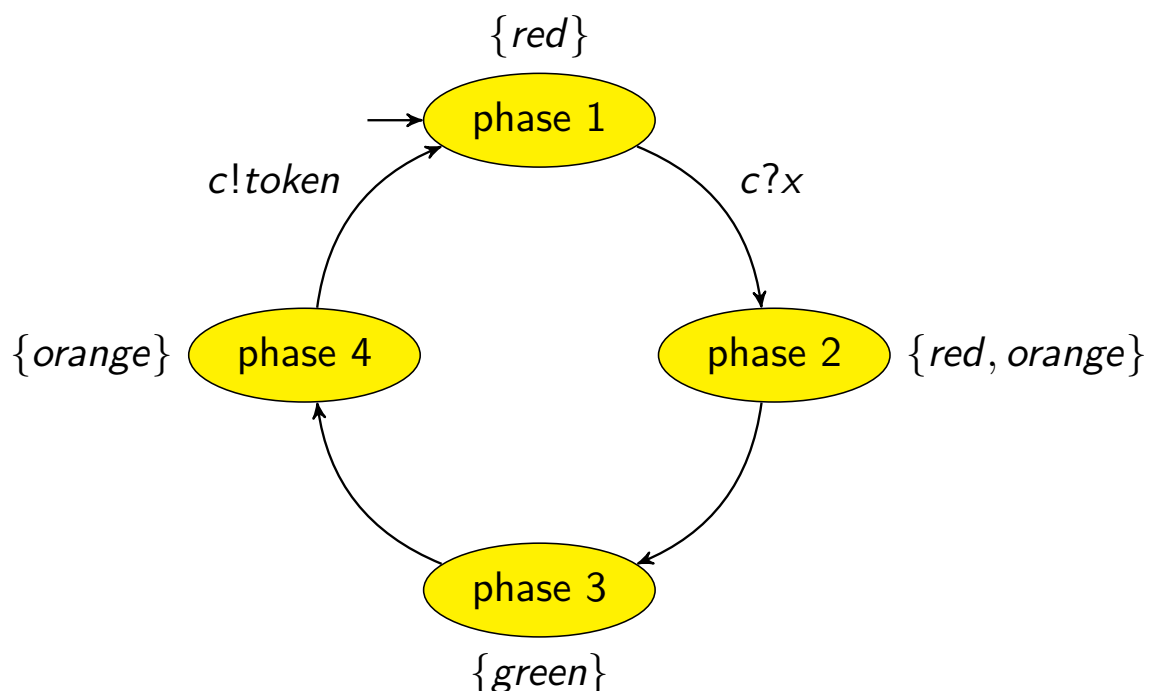
usually, processes exchange data in some way  $\Rightarrow$  **channels**

- 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 receiving
  - 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 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



say that channel only has one value: *token*, say that  $cap(c) = 0$

## 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  $(\bigcup_{0 < i \leq n} Var_i, Chan)$ :

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

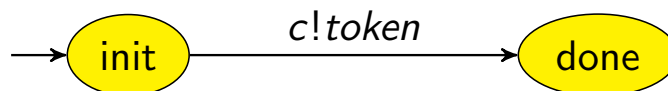
with program graphs  $PG_i$  over  $(Var_i, Chan)$

## Example: Traffic lights

NOT IN HANDOUT, GRAPH IN PRESENTATION

$$Crossing = [TrafficLight \mid TrafficLight \mid Starter]$$

- only two traffic lights  $\Rightarrow$  both wait for input infinitely long
- therefore use additional “starter” to send one input



## 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  $CS = [PG_1 \mid \dots \mid PG_n]$  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**  $(S, \rightarrow, I, AP, L)$  where:

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

# Inference rules (I)

- interleaving for  $\alpha \in Act_i$ :

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

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

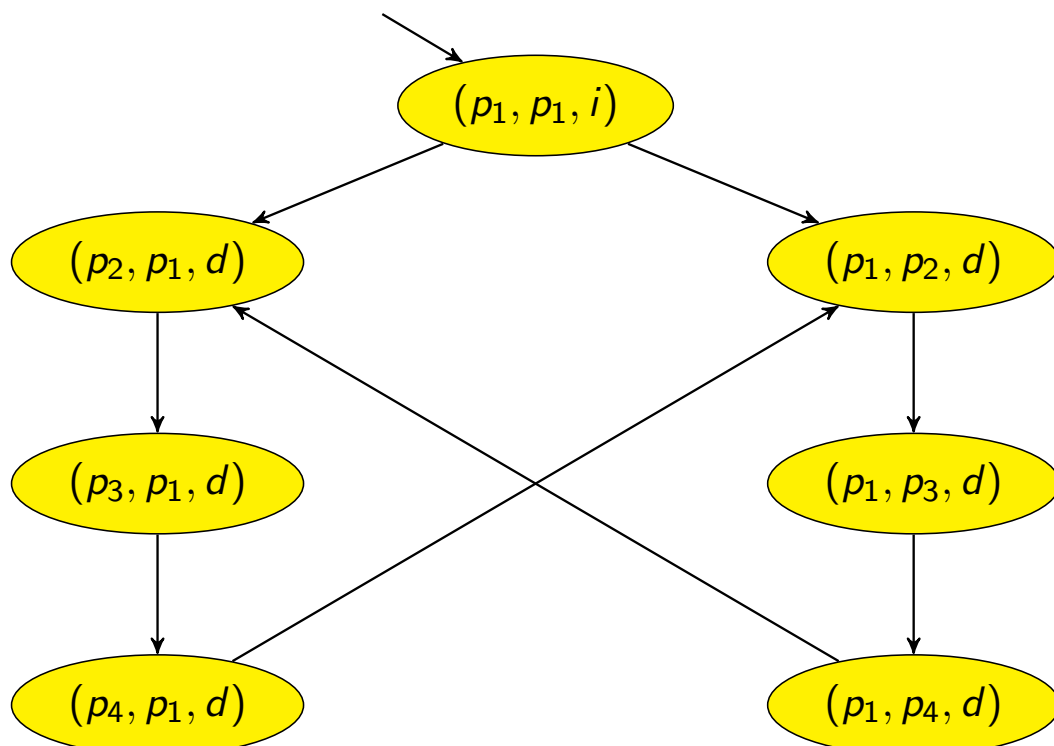
- synchronous message passing over  $c \in Chan$ ,  $cap(c) = 0$ :

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

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

## Example: Traffic lights

NOT IN HANDOUT, PROGRAM GRAPH IN PRESENTATION



mention unreachable states, no evaluation, no channel evaluations

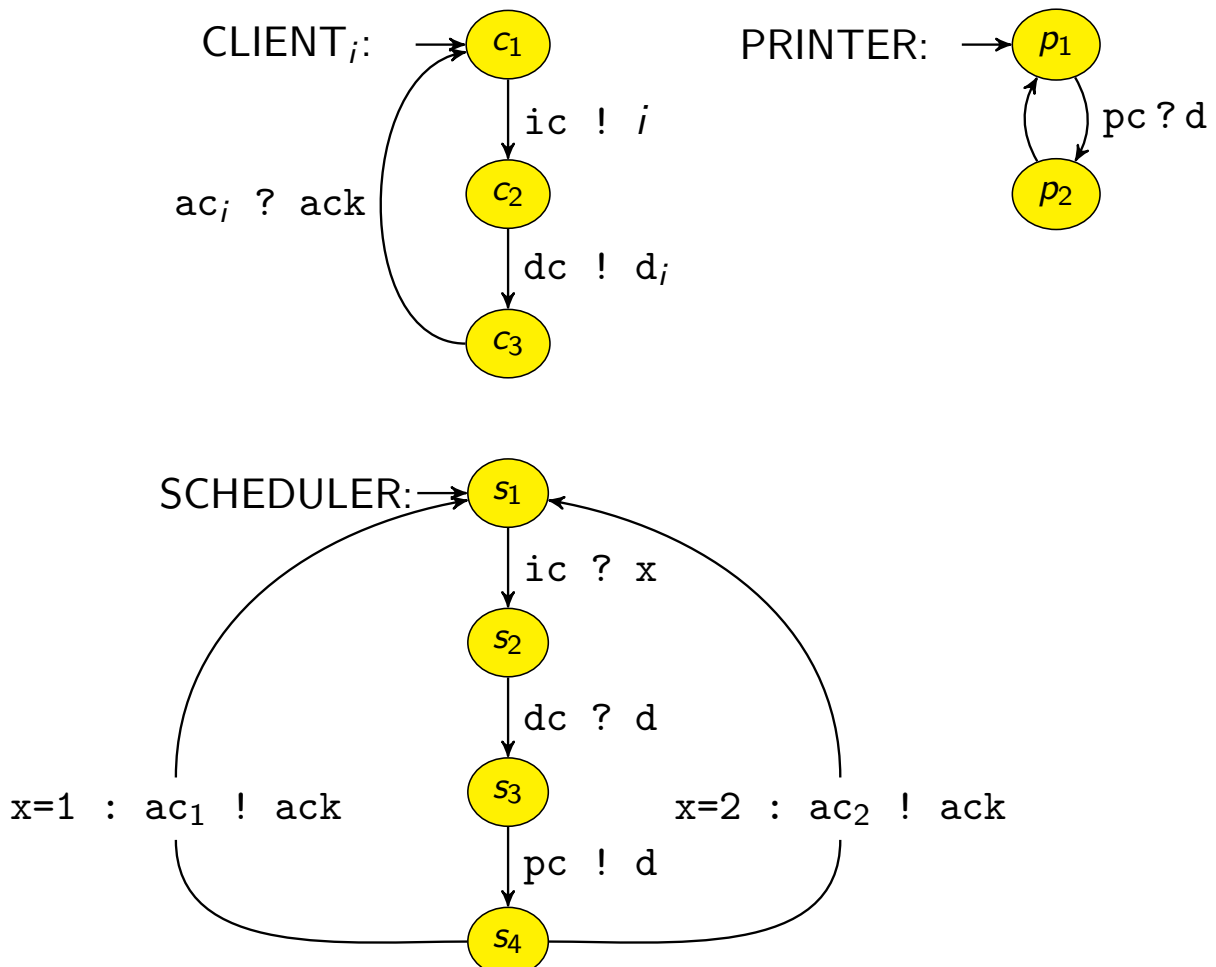
# Example protocol

- two clients, one scheduler, one printer
- clients send data to scheduler which sends this data further to printer
- before sending data, clients have to initialize connection by sending id
- after data has been delivered by printer, scheduler sends ack. to client
- scheduler should always be able to receive data

⇒ proper modeling requires **asynchronous message passing**

- if  $cap(c) > 0$ , then process  $P_i$  can perform  $l_i \xrightarrow{g:c!v} l'_i$
- ... iff  $g$  is satisfied and less than  $cap(c)$  messages are stored in  $c$
- $P_j$  may perform  $l_j \xrightarrow{g:c?x} l'_j$  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**  $\xi$  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  $x$ :

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

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

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

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

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

## Transition system of example protocol

let  $c(ac_1) = c(ac_2) = c(pc) = 0$  and  $c(ic) = c(dc) > 0$   
 CHANNEL SYSTEM SHOWN IN PRESENTATION

$CL_1$	$CL_2$	$SC$	$PR$	$x$	$SC.d$	$PR.d$	$ic$	$dc$
$c_1$	$c_1$	$s_1$	$p_1$	?	?	?	$\epsilon$	$\epsilon$
$c_2$	$c_1$	$s_1$	$p_1$	?	?	?	1	$\epsilon$
$c_2$	$c_1$	$s_2$	$p_1$	1	?	?	$\epsilon$	$\epsilon$
$c_2$	$c_2$	$s_2$	$p_1$	1	?	?	2	$\epsilon$
$c_2$	$c_3$	$s_2$	$p_1$	1	?	?	2	$d_2$
$c_2$	$c_3$	$s_3$	$p_1$	1	$d_2$	?	2	$\epsilon$
$c_2$	$c_3$	$s_4$	$p_2$	1	$d_2$	$d_2$	2	$\epsilon$
$c_3$	$c_3$	$s_4$	$p_2$	1	$d_2$	$d_2$	2	$d_1$
$c_1$	$c_3$	$s_1$	$p_2$	1	$d_2$	$d_2$	2	$d_1$

problem: client 1 gets acknowledge although data 2 is send to printer

## Outline

- Program Graphs
- Channel systems
- Promela
  - Promela - Syntax and Intuitive Meaning
  - Formal semantics
- The State-Space Explosion Problem



## 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 explicit 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**:

$$\begin{aligned} \text{stmt} & ::= \mathbf{skip} \mid x := \text{expr} \mid c?x \mid c!\text{expr} \mid \\ & \quad \text{stmt}_1 ; \text{stmt}_2 \mid \mathbf{atomic}\{\text{assignments}\} \mid \\ & \quad \mathbf{if} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \dots \quad :: g_n \Rightarrow \text{stmt}_n \quad \mathbf{fi} \quad \mid \\ & \quad \mathbf{do} \quad :: g_1 \Rightarrow \text{stmt}_1 \quad \dots \quad :: g_n \Rightarrow \text{stmt}_n \quad \mathbf{od} \\ \text{assignments} & ::= x_1 := \text{expr}_1 ; x_2 := \text{expr}_2 ; \dots ; x_m := \text{expr}_m \end{aligned}$$

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

## Conditional statements

$$\mathbf{if} \ :: g_1 \Rightarrow \text{stmt}_1 \ \dots \ :: g_n \Rightarrow \text{stmt}_n \ \mathbf{fi}$$

- nondeterministic choice between statements  $\text{stmt}_i$  for which  $g_i$  holds
- test-and-set semantics: (deviation from Promela)
  - guard evaluation + selection of enabled command + execution first atomic step of selected statement is all performed **atomically**
- **if-fi**-command **blocks** if no guard holds
  - parallel processes may unblock a process by changing shared variables
  - e.g., when  $y=0$ , **if**  $:: y > 0 \Rightarrow x := 42$  **fi** waits until  $y$  exceeds 0
- standard abbreviations:
  - **if**  $g$  **then**  $\text{stmt}_1$  **else**  $\text{stmt}_2$  **fi**  $\equiv$  **if**  $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow \text{stmt}_2$  **fi**
  - **if**  $g$  **then**  $\text{stmt}_1$  **fi**  $\equiv$  **if**  $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow$  **skip** **fi**

## Iteration statements

$$\mathbf{do} \ :: g_1 \Rightarrow \text{stmt}_1 \ \dots \ :: g_n \Rightarrow \text{stmt}_n \ \mathbf{od}$$

- iterative execution of nondeterministic choice among  $g_i \Rightarrow \text{stmt}_i$ 
  - where guard  $g_i$  holds in the current state
- no blocking if all guards are violated; instead, loop is aborted
- **while**  $g$  **do**  $\text{stmt}$  **od**  $\equiv$  **do**  $:: g \Rightarrow \text{stmt}$  **od**
- no break-statements to abort a loop

## Beverage vending machine

the following nanoPromela program describes its behaviour:

```
atomic { nbeer := max; nsprite := max };
do :: true => skip; // insert coin
    if :: nsprite > 0 => nsprite := nsprite - 1
        :: nbeer > 0 => nbeer := nbeer - 1
        :: nsprite = nbeer = 0 => skip
    fi
:: true => atomic { nbeer := max; nsprite := max }
od
```

## Client-scheduler-printer example

```
----- CLIENT i -----
do :: true => ic ! i;
    dc ! di;
    aci ? ack
od
----- SCHEDULER -----
do :: true => ic ? x;
    dc ? d;
    pc ! d;
    if :: x = 1 => ac1 ! ack
        :: x = 2 => ac2 ! ack
    fi
od
----- PRINTER -----
do :: true => pc ? d; skip
od
```

## Formal semantics

the **semantics** of a nanoPromela-statement over  $(Var, Chan)$  is a **program graph** over  $(Var, Chan)$ .

the program graphs  $PG_1, \dots, PG_n$  for the processes  $\mathcal{P}_1, \dots, \mathcal{P}_n$  of a nanoPromela-program  $\overline{\mathcal{P}} = [\mathcal{P}_1 | \dots | \mathcal{P}_n]$  constitute a **channel system** over  $(Var, Chan)$

the locations of the program graph  $PG_i$  are the **sub-statements** of the nanoPromela-program  $\mathcal{P}_i$

## Sub-statements

for statement **stmt** its **sub-statements**  $Sub(stmt)$  is **smallest set of statements** such that

- **exit**  $\in Sub(stmt)$
- **stmt**  $\in Sub(stmt)$
- if **stmt'**  $\in Sub(stmt)$  then  $Sub(stmt') \subseteq Sub(stmt)$
- if **stmt'**  $\in Sub(stmt_1)$  then **stmt'; stmt<sub>2</sub>**  $\in Sub(stmt_1; stmt_2)$
- if **stmt'**  $\in Sub(stmt_2)$  then **stmt'**  $\in Sub(stmt_1; stmt_2)$
- if **stmt'**  $\in Sub(stmt_i)$  then **stmt'**  $\in Sub(\mathbf{if} \dots :: g_i \Rightarrow stmt_i \dots \mathbf{fi})$
- if **stmt'**  $\in Sub(stmt_i)$  then **stmt'; loop**  $\in Sub(loop)$  where **loop** = **do**  $\dots :: g_i \Rightarrow stmt_i \dots$  **od**

## Inference rules

$$\frac{}{\text{skip} \rightarrow \text{exit}}$$

$$x := \text{expr} \xrightarrow{\text{assign}(x, \text{expr})} \text{exit}$$

$\text{assign}(x, \text{expr})$  denotes the action that only changes  $x$ , no other variables

$$c?x \xrightarrow{c?x} \text{exit}$$

$$c!\text{expr} \xrightarrow{c!\text{expr}} \text{exit}$$

## Inference rules

$$\frac{}{\text{atomic}\{x_1 := \text{expr}_1; \dots; x_m := \text{expr}_m\} \xrightarrow{\alpha_m} \text{exit}}$$

where  $\alpha_0 = id$ ,  $\alpha_i = \text{Effect}(\text{assign}(x_i, \text{expr}_i), \text{Effect}(\alpha_{i-1}, \eta))$  for  $1 \leq i \leq m$

$$\frac{\text{stmt}_1 \xrightarrow{g:\alpha} \text{stmt}'_1 \neq \text{exit}}{\text{stmt}_1; \text{stmt}_2 \xrightarrow{g:\alpha} \text{stmt}'_1; \text{stmt}_2}$$

$$\frac{\text{stmt}_1 \xrightarrow{g:\alpha} \text{exit}}{\text{stmt}_1; \text{stmt}_2 \xrightarrow{g:\alpha} \text{stmt}_2}$$

## Inference rules

$$\frac{\text{stmt}_j \xrightarrow{h:\alpha} \text{stmt}'_j}{\text{if } \dots :: g_j \Rightarrow \text{stmt}_j \dots \text{fi} \xrightarrow{g_j \wedge h:\alpha} \text{stmt}'_j}$$

$$\frac{\text{stmt}_j \xrightarrow{h:\alpha} \text{stmt}'_j \neq \text{exit}}{\text{do } \dots :: g_j \Rightarrow \text{stmt}_j \dots \text{od} \xrightarrow{g_j \wedge h:\alpha} \text{stmt}'_j; \text{do } \dots \text{od}}$$

$$\frac{\text{stmt}_j \xrightarrow{h:\alpha} \text{exit}}{\text{do } \dots :: g_j \Rightarrow \text{stmt}_j \dots \text{od} \xrightarrow{g_j \wedge h:\alpha} \text{do } \dots \text{od}}$$

$$\frac{}{\text{do } \dots :: g_j \Rightarrow \text{stmt}_j \dots \text{od} \xrightarrow{\neg g_1 \wedge \dots \wedge \neg g_n} \text{exit}}$$

## Example: one step

let *loop* be a shortcut for

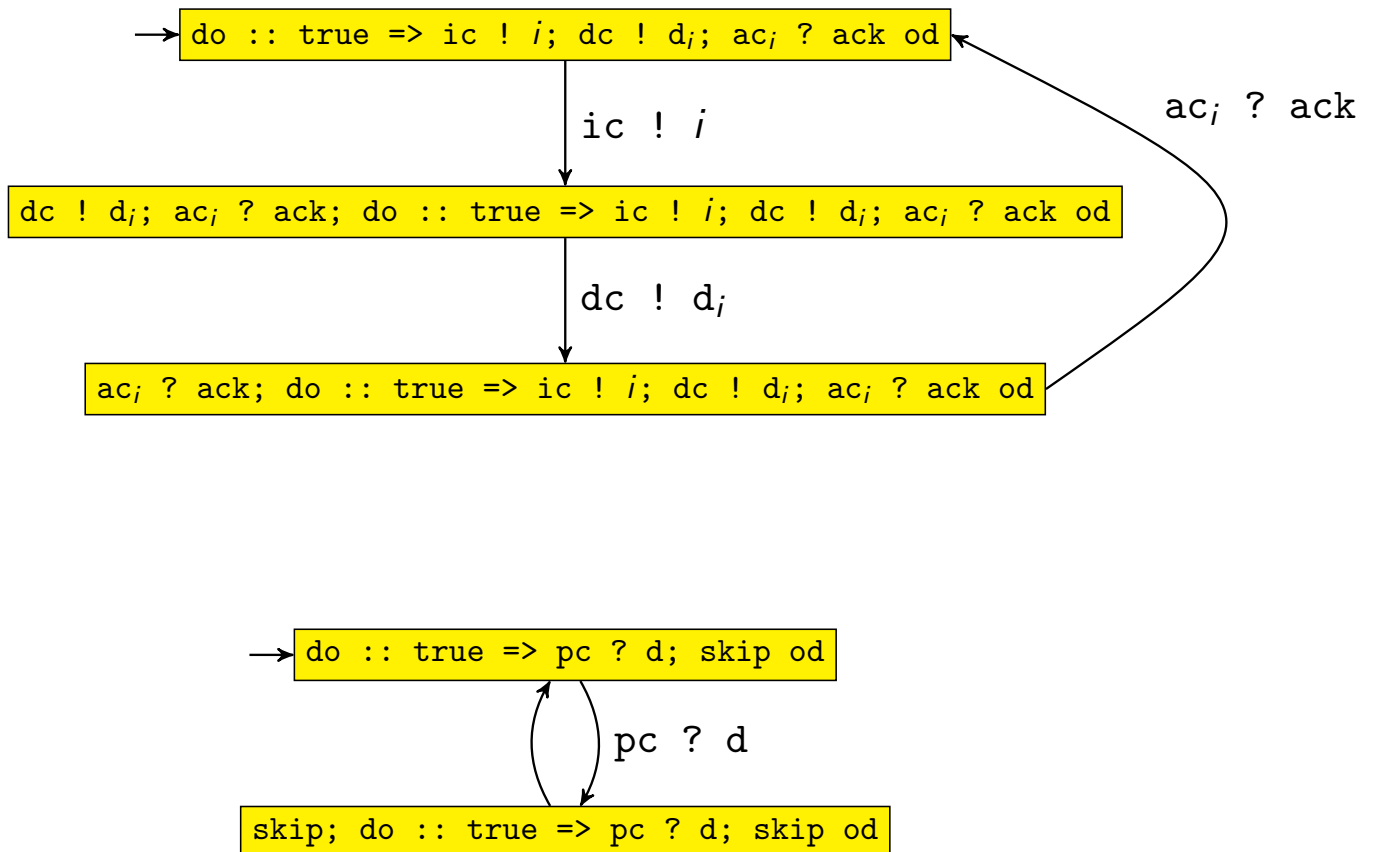
```
do :: true => ic ! i; dc ! d_j; ac_j ? ack od
```

derive the following step in the program graph of the client

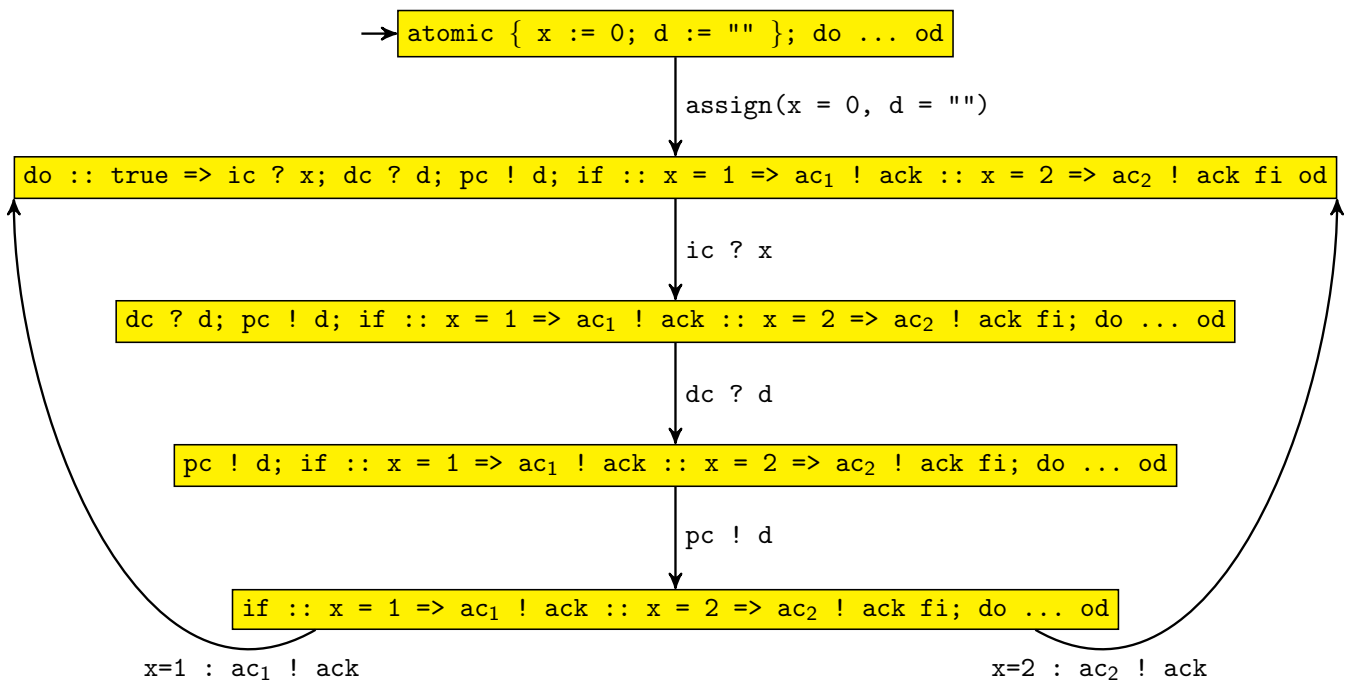
$$\frac{\frac{\text{ic ! } i \xrightarrow{\text{ic ! } i} \text{exit}}{\text{ic ! } i; \text{dc ! } d_j; \text{ac}_j ? \text{ack} \xrightarrow{\text{ic ! } i} \text{dc ! } d_j; \text{ac}_j ? \text{ack}}}{\text{loop} \xrightarrow{\text{ic ! } i} \text{dc ! } d_j; \text{ac}_j ? \text{ack}; \text{loop}}$$

construct by going left-down, left-up, right-up, right-down

## Example: client and printer



## Example: scheduler



# Outline

- Program Graphs
- Channel systems
- Promela
  - Promela - Syntax and Intuitive Meaning
  - Formal semantics
- The State-Space Explosion Problem

## Sequential programs

- # states of a simple program graph is:

$$|\text{\#program locations}| \cdot \prod_{\text{variable } x} |\text{dom}(x)|$$

- ⇒ number of states grows **exponentially** in number of program variables
    - $N$  variables with  $k$  possible values each yields  $k^N$  states
    - this is called the **state-space explosion problem**
- program with 10 locations, 3 bools, 4 integers (in range 0...9):

$$10 \cdot 2^3 \cdot 10^4 = 800,000 \text{ states}$$

- adding a single 50-positions bit-array yields  $800,000 \cdot 2^{50}$  states



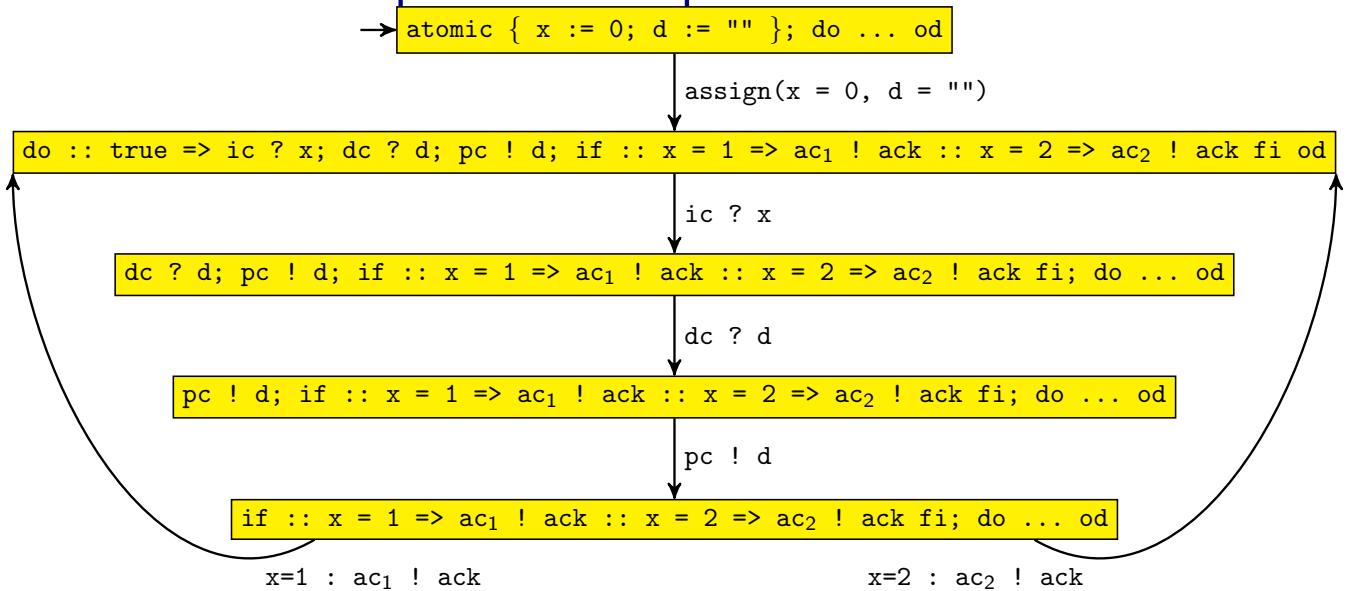
# Channel systems

- each channel  $c$  has a bounded capacity  $cap(c)$  and a domain  $dom(c)$
- # states of system with  $N$  components and  $K$  channels is:

$$\prod_{i=1}^N \left( |\# \text{program locations}| \prod_{\text{variable } x} |dom(x)| \right) \cdot \prod_{j=1}^K |dom(c_j)|^{cap(c_j)+1}$$

this is the underlying structure of Promela

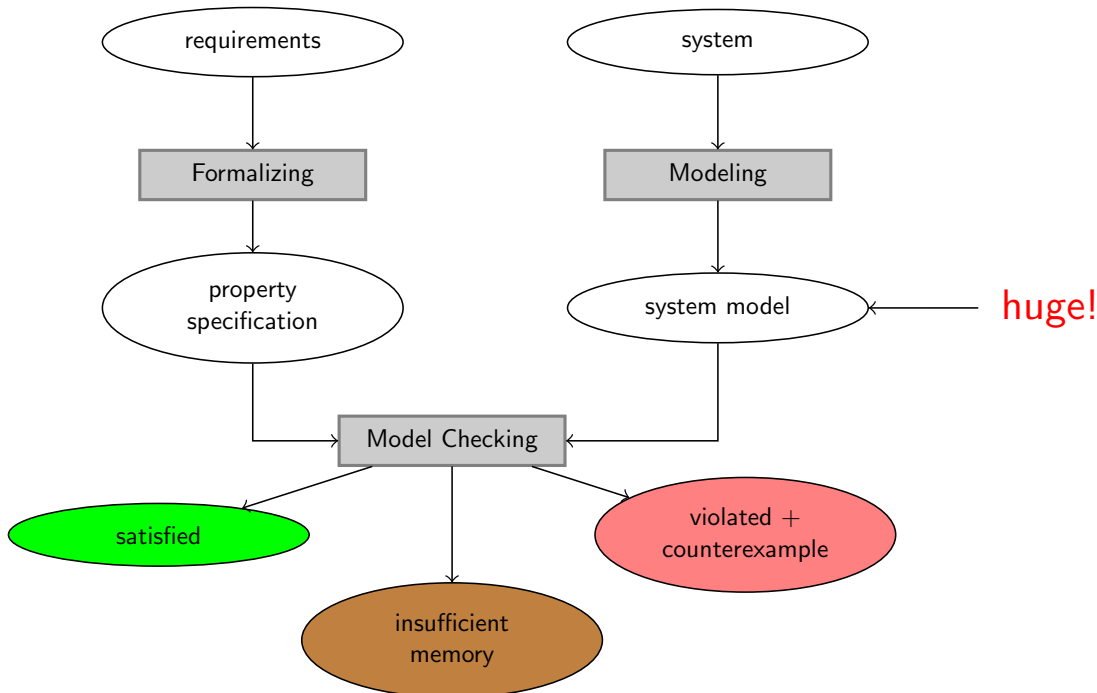
## Client-scheduler-printer example



for channel capacity 6 and binary data obtain

$$\underbrace{3}_{\text{client 1}} \cdot \underbrace{3}_{\text{client 2}} \cdot \underbrace{5 \cdot 2^2}_{\text{scheduler}} \cdot \underbrace{2 \cdot 2}_{\text{printer}} \cdot \underbrace{2^{6+1}}_{ic} \cdot \underbrace{2^{6+1}}_{dc} = 45 \cdot 2^{18} = 11,796,480 \text{ states}$$

# The Need for Automated Verification



## Summary of Modeling Concurrent Systems

- **transition systems** are fundamental for modeling software  
**should be generated** from high-level modeling language
- **program graphs** = states with variables
- **interleaving** = execution of independent concurrent processes by nondeterminism
- **channel systems** = program graphs + first-in first-out communication
  - handshaking for channels of capacity 0
  - asynchronous message passing when capacity exceeds 0
  - semantical model of **Promela**
- **formal semantic** for Promela  $\Rightarrow$  know exactly which system is verified
- size of transition systems grows **exponentially**
  - in the number of concurrent components, number of variables, and size of channels