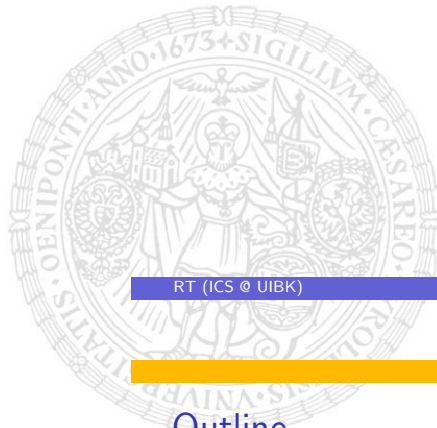


Introduction to Model Checking

René Thiemann

Institute of Computer Science
University of Innsbruck

WS 2007/2008



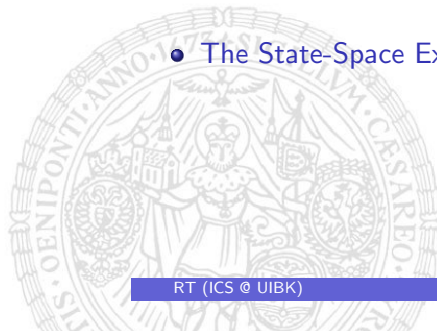
RT (ICS @ UIBK)

week 4

1/23

Outline

- Promela - Syntax and Intuitive Meaning
- Promela - Formal semantics
- The State-Space Explosion Problem



RT (ICS @ UIBK)

week 4

2/23

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:

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

x is a variable in *Var*, *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 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**
- The **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** stmt_1 **else** stmt_2 **fi** \equiv **if** $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow \text{stmt}_2$ **fi**
 - **if** g **then** stmt_1 **fi** \equiv **if** $:: g \Rightarrow \text{stmt}_1 \ :: \neg g \Rightarrow \text{skip}$ **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
- **do** $:: g \Rightarrow \text{stmt}$ **od** \equiv **while** g **do** stmt **od**
- No break-statements to abort a loop

Beverage vending machine

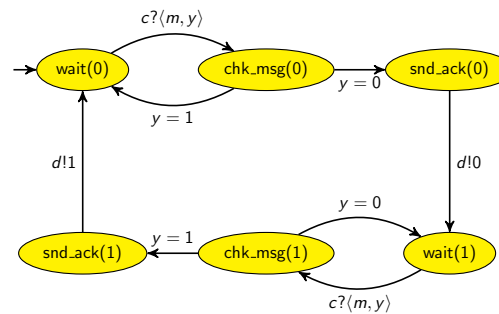
The following nanoPromela program describes its behaviour:

```

do :: true  $\Rightarrow$ 
    skip;
    if :: nsprite > 0  $\Rightarrow$  nsprite := nsprite - 1
        :: nbeer > 0  $\Rightarrow$  nbeer := nbeer - 1
        :: nsprite = nbeer = 0  $\Rightarrow$  skip
    fi
    :: true  $\Rightarrow$  atomic{nbeer := max; nsprite := max}
od

```

Alternating bit protocol - Receiver



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 $\bar{\mathcal{P}} = [\mathcal{P}_1 | \dots | \mathcal{P}_n]$ 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 each statement `stmt` its **substatements** $Sub_{\text{stmt}}(\text{stmt})$ is the **smallest set of statements** such that

Inference rules

$$\text{skip} \xrightarrow{\text{true: } id} \text{exit}$$

where id denotes an action that does not change the values of the variables

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

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

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

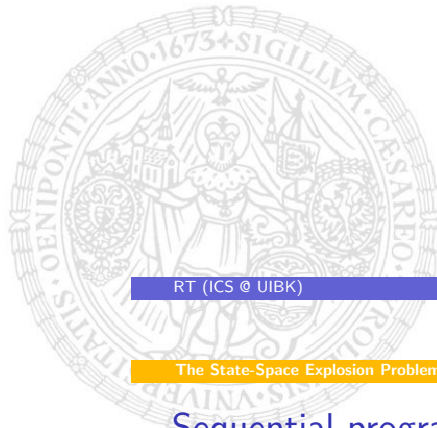
$$c! \text{expr} \xrightarrow{c! \text{expr}} \text{exit}$$

Inference rules

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

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

Inference rules



Sequential programs

- The # states of a simple program graph is:

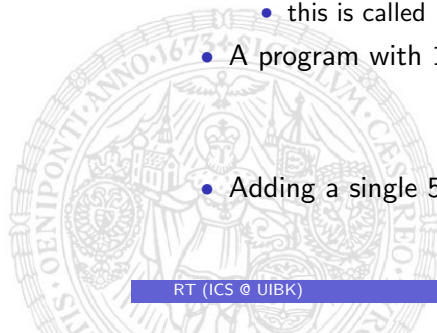
$$|\# \text{program locations}| \cdot \prod_{\text{variable } x} |\text{dom}(x)|$$

⇒ number of states grows *exponentially* in the number of program variables

- N variables with k possible values each yields k^N states
- this is called *the state-space explosion problem*
- A 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



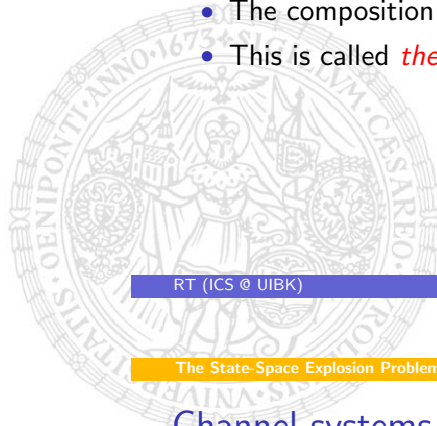
Concurrent programs

- The # states of $P \equiv P_1 \parallel \dots \parallel P_n$ is maximally:

$$\#states\ of\ P_1 \times \dots \times \#states\ of\ P_n$$

⇒ # states grows *exponentially* with the number of components

- The composition of N components of size k each yields k^N states
- This is called *the state-space explosion problem*

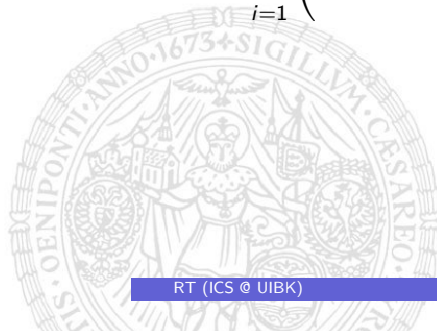


Channel systems

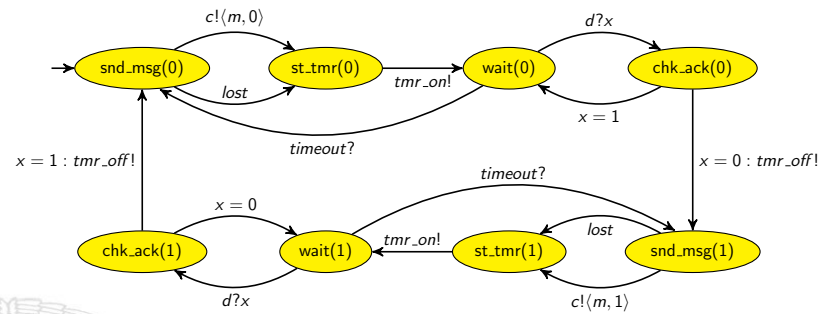
- Asynchronous communication of processes via *channels*
 - each channel c has a bounded capacity $cap(c)$
 - if a channel has capacity 0, we obtain *handshaking*
- # states of system with N components and K channels is:

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

this is the underlying structure of Promela



The alternating bit protocol

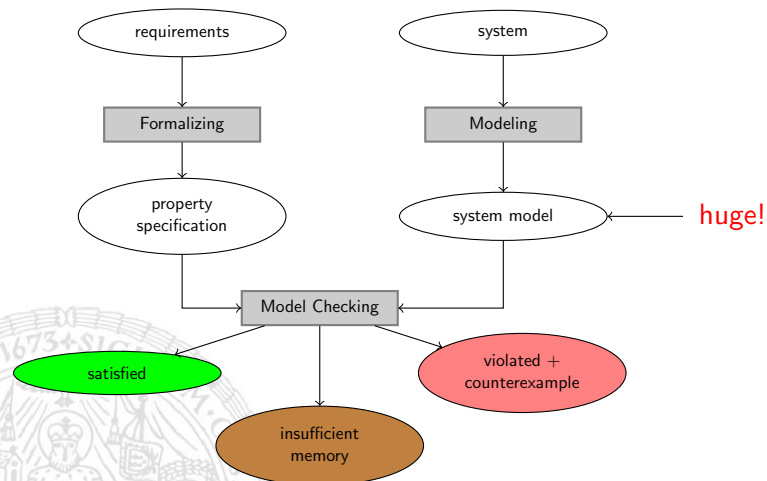


channel capacity 10, and datums are bits, yields

$$2 \cdot 8 \cdot 2 \cdot 6 \cdot 2 \cdot 4^{10} \cdot 2^{10} = 3 \cdot 2^{37} \approx 4 \cdot 10^{11} \text{ states}$$

timer
sender
rcv
c
d

The Need for Automated Verification



Summary of Modeling Concurrent Systems

- **Transition systems** are fundamental for modeling software
- **Interleaving** = execution of independent concurrent processes by nondeterminism
- **Channel systems** = program graphs + first-in first-out communication channels
 - 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 and the number of variables

Exercises

- Write promela-code for the timer and the sender of the alternating bit protocol.
- Construct the corresponding program graphs according to the formal semantics.
- Change the inference rules such that conditional- and iteration-statements require one step for choosing the case without partially evaluating the corresponding statement.
(Note that this refinement corresponds to the semantics of full Promela.)