# Formal Methods and Functional Programming

## Übungsstunde 13: Modeling

Department of Computer Science, ETH Zürich

Robin Ebersberger, LEE C 114

21.05.2025

# Überblick

- Recap Vorlesung
- Modeling in Promela
- HS22: Promela Aufgabe
- Leader Election Protocol

# Recap der Vorlesung

# Warum Modeling?

- NS & Hoare Logic:
  - keine Aussagen über Intermediate States
  - keine Aussagen bei Non-Termination
- Beweise in SOS umständlich
  - über alle möglichen Derivation Sequences (parallelism, non-determinism)
  - Non-Termination: braucht Induction Hypothesis (nicht automatisierbar)
- Explicit Model Checking
  - Enumerate all possible states of a system
  - State represented as concrete values

# Promela & Spin

- Promela = Protocol Meta Language
- Spin = Model Checker
  - simulates (executes) models written in Promela
- Slides Vorlesung: vollständige Erklärungen

# Promela Types

- `bit`, `bool`: $0 \ldots 1$
- `byte`: $0 \ldots 255$
- `short`: $-2^{15} \ldots 2^{15} - 1$
- `int`: $-2^{31} \ldots 2^{31} - 1$

- Arrays: `int name[4]`
- Structs
- `mtype`: symbolic constants
  - `mtype = {req, ack, err}`

- Channel type: `chan`

# Promela Basics

- Variable Declarations (Local/Global)
  - `byte a = 5;`
  - `vector v;`
  - initialized to zero-equivalent values
- Channel Declarations (Local/Global)
  - `chan c1 = [2] of {type1, type2, ...}`
  - `chan c2 = [0] of {int}`
  - msg buffer, then msg type
- `assert(E)`
- Expression Statement
  - `x > 0;` ->Blockt, bis Expression zu Wert ≠0 auswertet

# Promela Statements: Selection

- für if/else oder [] aus IMP

```
if
:: s1   /* option 1 */
:: ...
:: sn   /* option n */
fi
```

- Executable if at least one of its options it executable
- Chooses an option non-deterministically and executes it

```
if  /* Move a sprite */
:: x < maxX -> x = x + 1;
:: x > minX -> x = x - 1;
:: y < maxY -> y = y + 1;
:: y > minY -> y = y - 1;
:: color = color + 1;
fi
```

- Statement `else` is executable if no other option is executable (may occur at most in one option)

# Promela Statements: Repetition

- für Loops

```
do
:: s1   /* option 1 */
:: ...
:: sn   /* option n */
od
```

- Executable if at least one of its options it executable
- Chooses repeatedly an option non-deterministically and executes it
- Terminates when a `break` or `goto` is executed

```
/* compute factorial of n */

int r = 1;

do
:: n > 1 -> r = r*n; n = n-1;
:: else -> break
od
```

# Promela Atomicity

- Basic Statements executed atomically
  - Model Java `x = x + 1` as `y = x; y = y + 1; x = y`
- Atomic Blocks
  - Executed atomically sobald ausführbar
  - Falls ein späteres Stmt im atomic {...} blockt, geht atomicity verloren

```
/* lock */
atomic {
  locked == 0;
  locked = 1;
}
/* critical section */
locked = 0; /* unlock */
```

# Promela Macros

- keine Methoden, dafür Makros
- Schlichte Ersetzung im Code
  - Vorsicht mit Scope von Variablen
  - keine Rekursion

```
inline lock() {
  atomic {
    locked == 0;
    locked = 1
  }
}
```

```
inline swap(a, b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp
}
```

# Promela Processes

- `init {...}`
  - “Main-Process”
- `proctype myProcess(int p) {...}`
  - müssen von laufendem Prozess gestartet werden
  - `run myProcess(5);`
- `active [N] proctype myProcess(...) {...}`
  - Startet N dieser Prozesse direkt am Anfang
- `_nr_pr` ist Anzahl aktiver Prozesse
  - `_nr_pr == 1;`
  - blockt, bis es nur noch einen Prozess (diesen) gibt

# Promela Channels

- `chan ch = [d] of {type1, ..., typeN}`
- `ch ! e1, ..., eN`
  - sends Msg on Channel ch
  - blockt, falls Buffer voll oder unbuffered
- `ch ? a1, ..., aN`
  - receives Msg from Channel ch
  - `a1, ..., aN` sind Vars, Konstanten oder _
  - blockt, falls keine Msg da oder Konstanten nicht gematcht

# State Space Explosions

- Anzahl States des Programms wächst exponentiell in
  - der Anzahl Variablen
  - der Anzahl Prozesse (Threads)
  - der Anzahl & Kapazität von Channels

# Aufgaben

# Spin - Installation

- Windows: spinroot.com/spin/Man/README.html
- Ubuntu: Run `sudo apt-get install spin` in a terminal
- Mac: Use Homebrew (https://brew.sh) to install Spin by running `brew install spin`.
  - Homebrew insgesamt Empfehlung

# Spin - Getting Started

- `spin model.pml` für *eine* Simulation
- `spin -a model.pml` generiert ein C-Programm (`pan.c`) für einen vollständigen Model-Check
- C-Progamm mit `gcc pan.c -o tester && ./tester` compilen und ausführen
- Gegenbeispiele werden automatisch gespeichert, diese mit `spin -t model.pml` ausführen