Applications of Formal Verification
Model Checking with Temporal Logic

Prof. Dr. Bernhard Beckert · Dr. Vladimir Klebanov | SS 2010
Model Checking with SPIN

- **model name.pml**
- **correctness properties**

**SPIN** -> **verifier pan.c**

**C compiler** -> **executable verifier pan**

**random/interactive/guided simulation**

- failing run **name.pml.trail**
  - either
  - “errors: 0”
Correctness properties can be stated syntactically within or outside the model.

**Stating properties within model using**
- assertion statements
- meta labels
  - end labels
  - accept labels
  - progress labels

**Stating properties outside model using**
- never claims
- *temporal logic formulas* (today’s main topic)
many correctness properties not expressible by assertions today:

model checking of properties formulated in temporal logic

Remark:
in this course, “temporal logic” is synonymous to “linear temporal logic” \((LTL)\)
Beyond Assertions

Assertions only talk about the state ‘at their own location’ in the code.

Example: mutual exclusion expressed by adding assertion into each critical section.

```c
  critical++;  
  assert( critical <= 1 );  
  critical--;  
```

Drawbacks:

- no separation of concerns (model vs. correctness property)
- changing assertions is error prone (easily out of synch)
- easy to forget assertions: correctness property might be violated at unexpected locations
- many interesting properties not expressible via assertions
Temporal Correctness Properties

properties more conveniently expressed as global properties, rather than assertions:

Mutual Exclusion

"critical <= 1  holds throughout the run"

Array Index within Bounds (given array \( a \) of length \( \text{len} \))

"0 <= i <= \text{len}-1  holds throughout the run"

properties impossible to express via assertions:

Absence of Deadlock

"If some processes try to enter their critical section, eventually one of them does so."

Absence of Starvation

"If one process tries to enter its critical section, eventually that process does so."

all these are temporal properties  \( \Rightarrow \) use temporal logic
Boolean Temporal Logic

talking about numerical variables (like in \texttt{critical} \leq 1 \texttt{or} \texttt{0} \leq \texttt{i} \leq \texttt{len}-1) requires variation of \textit{propositional temporal logic} which we call \textit{Boolean temporal logic}:

- \textbf{Boolean expressions} (over \texttt{PROMELA} variables), rather than \textit{propositions}, form basic building blocks of the logic
Boolean Temporal Logic over PROMELA

Set \( \text{For}_{BTL} \) of Boolean Temporal Formulas (simplified)

- All PROMELA variables and constants of type \text{bool}/\text{bit} are in \( \text{For}_{BTL} \).
- If \( e_1 \) and \( e_2 \) are numerical PROMELA expressions, then all of \( e_1==e_2, e_1!=e_2, e_1<e_2, e_1<=e_2, e_1>e_2, e_1>=e_2 \) are in \( \text{For}_{BTL} \).
- If \( P \) is a process and \( l \) is a label in \( P \), then \( P@l \) is in \( \text{For}_{BTL} \) (\( P@l \) reads “\( P \) is at \( l \)”)
- If \( \phi \) and \( \psi \) are formulas in \( \text{For}_{BTL} \), then all of
  \[ \neg \phi, \quad \phi \land \psi, \quad \phi \lor \psi, \quad \phi \rightarrow \psi, \quad \phi \leftrightarrow \psi \]
  \[ []\phi, \quad <>\phi, \quad \phi \cup \psi \]
  are in \( \text{For}_{BTL} \).
A run $\sigma$ through a PROMELA model $M$ is a chain of states

$\mathcal{L}_0, \mathcal{I}_0 \rightarrow \mathcal{L}_1, \mathcal{I}_1 \rightarrow \mathcal{L}_2, \mathcal{I}_2 \rightarrow \mathcal{L}_3, \mathcal{I}_3 \rightarrow \mathcal{L}_4, \mathcal{I}_4 \rightarrow \ldots$

$\mathcal{L}_j$ maps each running process to its current location counter. From $\mathcal{L}_j$ to $\mathcal{L}_{j+1}$, only one of the location counters has advanced (exception: channel rendezvous).

$\mathcal{I}_j$ maps each variable in $M$ to its current value.

Arithmetic and relational expressions are interpreted in states as expected; e.g. $\mathcal{L}_j, \mathcal{I}_j \models x < y$ iff $\mathcal{I}_j(x) < \mathcal{I}_j(y)$

$\mathcal{L}_j, \mathcal{I}_j \models \mathcal{P}@1$ iff $\mathcal{L}_j(\mathcal{P})$ is the location labeled with 1

Evaluating other formulas $\in \text{For}_{BTL}$ in runs $\sigma$: see lecture 2.
SPIN supports Boolean temporal logic

but

arithmetic operators (+, -, *, /, ...),
relational operators (==, !=, <, <=, ...),
label operators (@)
cannot appear directly in TL formulas given to SPIN

instead

Boolean expressions must be abbreviated using \#define
Safety Properties

formulas of the form $[]\phi$ are called safety properties state that something good, $\phi$, is guaranteed throughout each run special case: $[]\neg\psi$ states that something bad, $\psi$, never happens

dexample: ‘$[](\text{critical} \leq 1)$’

“it is guaranteed throughout each run that at most one process visits its critical section”

or equivalently: “more than one process visiting its critical section will never happen”
Applying Temporal Logic to Critical Section Problem

We want to verify ‘[](critical<=1)’ as correctness property of:

```c
active proctype P() {
    do ::  /* non-critical activity */
        atomic {
        !inCriticalQ;
        inCriticalP = true
    }
    critical++;
    /* critical activity */
    critical--;  
inCriticalP = false
        od
    }
/* similarly for process Q */
```
1. add ‘#define mutex (critical <= 1)’ to PROMELA file
2. open PROMELA file
3. enter \([\text{mutex}])\) in LTL text field
4. select Translate to create a ‘never claim’, corresponding to the negation of the formula
5. ensure Safety is selected
6. select Verify
7. (if necessary) select Stop to terminate too long verification
you may ignore them, but if you are interested:

- a never claim tries to show the user wrong
- it defines, in terms of PROMELA, all violations of a wanted correctness property
- it is semantically equivalent to the negation of the wanted correctness property
- JSPIN adds the negation for you
- using SPIN directly, you have to add the negation yourself
Model Checking a Safety Property with **SPIN** directly

---

**Command Line Execution**

```make
make sure ‘#define mutex (critical <= 1)’ is in safety1.pml

> spin -a -f !([] mutex) safety1.pml
> gcc -DSAFETY -o pan pan.c
> ./pan
```
Temporal MC Without Ghost Variables

We want to verify mutual exclusion without using ghost variables

```c
#define mutex !(P@cs && Q@cs)
bool inCriticalP = false, inCriticalQ = false;

active proctype P() {
  do :: atomic {
    !inCriticalQ;
    inCriticalP = true
  }
  cs: /* critical activity */
  inCriticalP = false
  od
}
/* similarly for process Q */
/* with same label cs: */

Verify ‘[ ]mutex’ with JSPIN.
```
Liveness Properties

formulas of the form $<>\phi$ are called liveness properties
state that something good, $\phi$, eventually happens in each run

definition: $<>csp$

(example: ‘$<>csp$’
(with $csp$ a variable only true in the critical section of $P$)

“in each run, process $P$ visits its critical section eventually”
Applying Temporal Logic to Starvation Problem

We want to verify ‘<>csp’ as correctness property of:

```c
active proctype P() {
    do :: /* non-critical activity */
        atomic {
            !inCriticalQ;
            inCriticalP = true
        }
    csp = true;
    /* critical activity */
    csp = false;
    inCriticalP = false
    od
}
/* similarly for process Q */
/* here using csq */
```
Model Checking a Liveness Property with JSpin

1. open PROMELA file
2. enter $<>\text{csp}$ in LTL text field
3. select Translate to create a ‘never claim’, corresponding to the negation of the formula
4. ensure that Acceptance is selected (SPIN will search for accepting cycles through the never claim)
5. for the moment uncheck Weak Fairness (see discussion below)
6. select Verify
Verification fails.

Why?

The liveness property on one process ‘had no chance’. Not even weak fairness was switched on!
Fairness

Does the following PROMELA model necessarily terminate?

```pascal
byte n = 0;
bool flag = false;

active proctype P() {
    do :: flag -> break;
    :: else -> n = 5 - n;
    od
}

active proctype Q() {
    flag = true
}

Termination guaranteed only if scheduling is (weakly) fair!

Definition (Weak Fairness)

A run is called weakly fair iff the following holds:
each continuously executable statement is executed eventually.
Always switch **Weak Fairness** on when checking for liveness!

1. open PROMELA file
2. enter \texttt{<>csp} in LTL text field
3. select Translate to create a ‘never claim’, corresponding to the negation of the formula
4. ensure that **Acceptance** is selected (SPIN will search for *accepting* cycles through the never claim)
5. ensure **Weak Fairness** is checked
6. select Verify
Model Checking Liveness with Spin directly

Command Line Execution

```bash
> spin -a -f "!csp" liveness1.pml
> gcc -o pan pan.c
> ./pan -a -f
```
Verification fails again.

Why?

Weak fairness is still too weak.

Note that \texttt{inCriticalQ} is \textbf{not} continuously executable!

Designing a fair mutual exclusion algorithm is complicated.
Literature for this Lecture

Ben-Ari Chapter 5