Model Checking with Temporal Logic

Srinivas Pinisetty [1]

April 2024

[1] Based on material from Prof. Wolfgang Ahrendt.

Model Checking

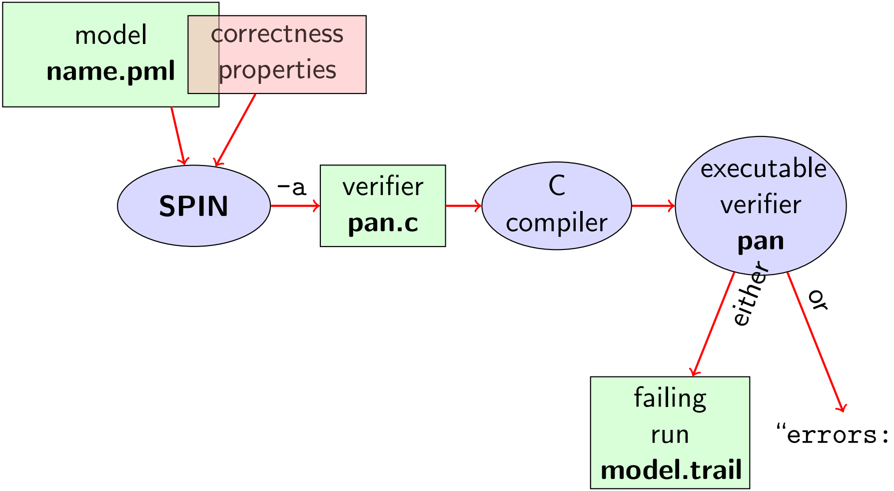
Check whether a formula is valid in all runs of a transition system.

Given a transition system $\mathcal{T}$ (e.g., derived from a PROMELA program).

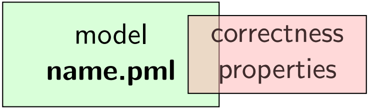
Verification task: is the LTL formula ϕ satisfied in all traces of $\mathcal{T}$, i.e.,

$$\mathcal{T} \models \phi \quad ?$$

Model Checking with SPIN

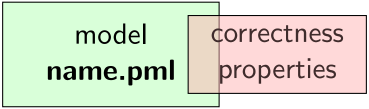


Stating Correctness Properties



Correctness properties can be stated within, or outside, the model.

Stating Correctness Properties

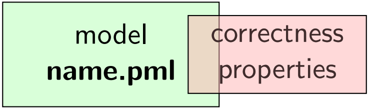


Correctness properties can be stated within, or outside, the model.

stating properties within model using

- assertion statements ✔

Stating Correctness Properties



Correctness properties can be stated within, or outside, the model.

stating properties within model using

- assertion statements ✔

stating properties outside model using

- temporal logic formulas

Preliminaries: Fairness

Does this model terminate in each run?

Simulate: `start/fair.pml`

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

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

Preliminaries: Fairness

Does this model terminate in each run?

Simulate: `start/fair.pml`

```
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!

Preliminaries: Fairness

Does this model terminate in each run?

Simulate: `start/fair.pml`

```
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.

Model Checking of Temporal Properties

Many correctness properties not expressible by assertions

- All properties that involve state changes
- Temporal logic expressive enough to characterize many (but not all) LT properties

In this course: "temporal logic" synonymous with "linear temporal logic"

Today: model checking of properties formulated in temporal logic

Beyond Assertions

Locality of Assertions

Assertions talk only about the state at their location in the code

Beyond Assertions

Locality of Assertions

Assertions talk only about the state at their location in the code

Example

Mutual exclusion enforced by adding assertion to each critical section

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

Beyond Assertions

Locality of Assertions

Assertions talk only about the state at their location in the code

Example

Mutual exclusion enforced by adding assertion to each critical section

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

Drawbacks

- No separation of concerns (model vs. correctness property)
- Changing assertions is error prone (easily out of sync)
- Easy to forget assertions:
  correctness property might be violated at unexpected locations

Beyond Assertions

Locality of Assertions

Assertions talk only about the state at their location in the code

Example

Mutual exclusion enforced by adding assertion to each critical section

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

Drawbacks

- No separation of concerns (model vs. correctness property)
- Changing assertions is error prone (easily out of sync)
- Easy to forget assertions:
  correctness property might be violated at unexpected locations
- Many interesting properties not expressible via assertions

Safety Properties

Safety Properties

- state that something ‘good’ is guaranteed throughout each run
- each violating run violates the property after *finitely* many steps

Safety Properties

Safety Properties

- state that something ‘good’ is guaranteed throughout each run
- each violating run violates the property after *finitely* many steps

Example

TL formula `[](critical <= 1)`

“It is guaranteed throughout each run that at most one process visits its critical section at any time.”

Safety Properties

Safety Properties

- state that something 'good' is guaranteed throughout each run
- each violating run violates the property after *finitely* many steps

Example

TL formula `[](critical <= 1)`

"It is guaranteed throughout each run that at most one process visits its critical section at any time."

or, equivalently:
"It will never happen that more than one process visits its critical section."

Applying Temporal Logic to Critical Section Problem

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

```
active proctype P() {
  do :: /* non-critical activity */
        atomic {
          !inCriticalQ;
          inCriticalP = true
        }
        critical++;
        /* critical activity */
        critical--;
        inCriticalP = false
  od
}

/* similarly for process Q */
```

Model Checking a Safety Property with SPIN

Command Line Execution

Add definition of TL formula to PROMELA file

Example `ltl atMostOne { [](critical <= 1) }`

General `ltl` *name* `{` *TL-formula* `}`

can define more than one formula

```
> spin -a file.pml
> gcc  -o pan pan.c
> ./pan -N name
```

Demo: `target/safety1.pml`

Model Checking a Safety Property with SPIN

Command Line Execution

Add definition of TL formula to PROMELA file

Example `ltl atMostOne { [](critical <= 1) }`

General `ltl` *name* `{` *TL-formula* `}`

can define more than one formula

```
> spin -a file.pml
> gcc  -o pan pan.c
> ./pan -N name
```

Demo: `target/safety1.pml`

Liveness Properties

Liveness Properties

- state that something good (ϕ) eventually happens in each run
- each violating requires *infinitely* many steps

Liveness Properties

Liveness Properties

- state that something good (ϕ) eventually happens in each run
- each violating requires *infinitely* many steps

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 a correctness property of:

```
active proctype P() {
  do :: /* non-critical activity */
        atomic {
          !inCriticalQ;
          inCriticalP = true
        }
        csp = true;
        /* critical activity */
        csp = false;
        inCriticalP = false
  od
}

/* similarly for process Q */
/* there, using csq         */
```

Model Checking a Liveness Property

1. open PROMELA file `liveness1.pml`
2. write `ltl pWillEnterC { <>csp }` in PROMELA file (as first `ltl` formula)
3. ensure that `Acceptance` is selected (SPIN will search for *accepting* cycles)
4. *for the moment* uncheck `Weak Fairness` (see discussion below)
5. select `Verify`

Verification Fails

Demo: `start/liveness1.pml`

Verification fails!

Why?

Verification Fails

Demo: `start/liveness1.pml`

Verification fails!

Why?

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

Model Checking Liveness with Weak Fairness

Always check `Weak fairness` when verifying liveness

1. open PROMELA file
2. write `ltl pWillEnterC { <>csp }` in PROMELA file
   (as first `ltl` formula)
3. ensure that `Acceptance` is selected
   (SPIN will search for *accepting* cycles through the never claim)
4. ensure `Weak fairness` is checked
5. select `Verify`

Model Checking Liveness using SPIN directly

Command Line Execution

Make sure `ltl` *name* { *TL-formula* } is in *file*.pml

```
> spin -a file.pml
> gcc -o pan pan.c
> ./pan -a -f [-N name]
```

`-a` acceptance cycles, `-f` weak fairness

Demo: `start/liveness1.pml`

Limitation of Weak Fairness

Verification fails again!

Why?

Limitation of Weak Fairness

Verification fails again!

Why?

Weak fairness is too weak . . .

Definition (Weak Fairness)

A run is called weakly fair iff the following holds:
each continuously executable statement is executed eventually.

Limitation of Weak Fairness

Verification fails again!

Why?

Weak fairness is too weak ...

Definition (Weak Fairness)

A run is called weakly fair iff the following holds:
each continuously executable statement is executed eventually.

Note that `!inCriticalQ` is not continuously executable!

Limitation of Weak Fairness

Verification fails again!

Why?

Weak fairness is too weak …

Definition (Weak Fairness)

A run is called weakly fair iff the following holds:
each continuously executable statement is executed eventually.

Note that `!inCriticalQ` is not continuously executable!

Restriction to weak fairness is principal limitation of SPIN

Here, liveness needs strong fairness, which is not supported by SPIN.

Revisit `fair.pml`

- Specify liveness of `fair.pml` using labels

Revisit `fair.pml`

- Specify liveness of `fair.pml` using labels
- Prove termination

Demo: `target/fair.pml`

Revisit `fair.pml`

- Specify liveness of `fair.pml` using labels
- Prove termination

Demo: `target/fair.pml`

- Here, weak fairness is needed, *and sufficient*

Literature for this Lecture

Ben-Ari (Principles of the Spin Model Checker) Chapter 5