

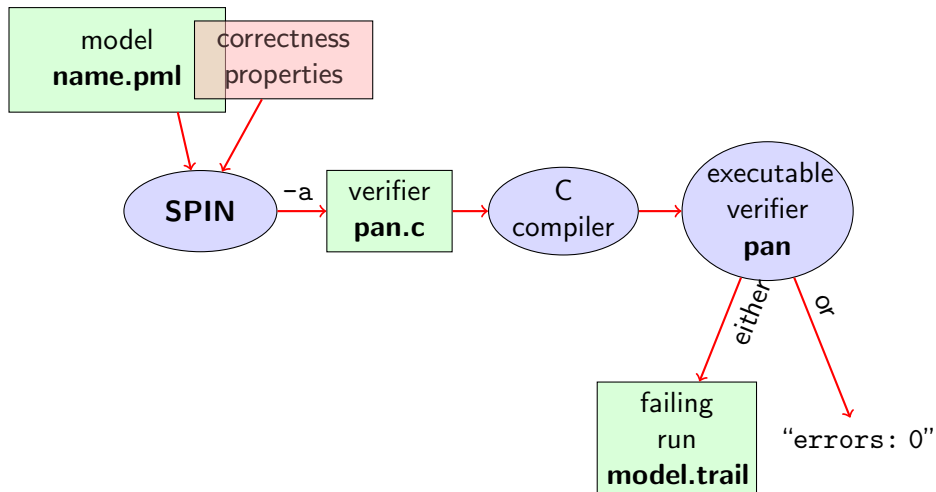
# Formal Methods for Software Development

## Model Checking with Temporal Logic

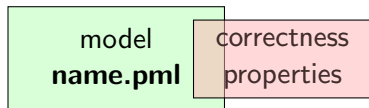
Wolfgang Ahrendt

18th September 2020

# Model Checking with SPIN



# Stating Correctness Properties



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

**stating properties within model** using

- ▶ assertion statements ✓
- ▶ meta labels
  - ▶ end labels ✓
  - ▶ **accept labels** (briefly)
  - ▶ progress labels

**stating properties outside model** using

- ▶ **never claims** (briefly)
- ▶ **temporal logic formulas** (today's main topic)

1. Accept labels in PROMELA  $\leftrightarrow$  Büchi automata
2. Fairness

# Preliminaries 1: Acceptance Cycles

## Definition (Accept Location)

A location marked with an **accept label** of the form “acceptxxx:” is called an **accept location**.

Accept locations can be used to **specify cyclic behavior**

## Definition (Acceptance Cycle)

A run which **infinitely often** passes through an **accept location** is called an **acceptance cycle**.

Acceptance cycles are mainly used in **never claims** (see below), to define (undesired) infinite behavior

## Preliminaries 2: 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) Linear Time 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

### 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:  
safety property might be violated at unexpected locations
- ▶ **Many interesting properties not expressible via assertions**



# Temporal Correctness Properties

Examples of properties where assertions are **suboptimal** (too local):

**Something should hold throughout**

“critical  $\leq 1$  holds **throughout** each run”

Examples of properties **impossible** to express as assertions:

**Something will hold eventually, infinitely often, ...**

“The traffic light will turn green infinitely often”

These are temporal properties  $\Rightarrow$  **use temporal logic**

## Numerical variables in expressions

- ▶ Expressions such as  $i \leq \text{len}-1$  contain numerical variables
- ▶ Propositional LTL as introduced so far only knows propositions
- ▶ Slight generalisation of LTL required

In **Boolean Temporal Logic**, atomic building blocks are  
*Boolean expressions* over PROMELA variables

# Boolean Temporal Logic over PROMELA

## Set $For_{BTL}$ of Boolean Temporal Formulas (simplified)

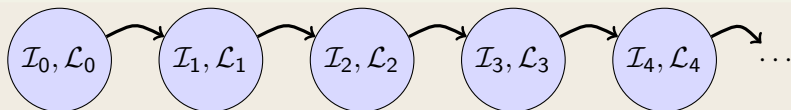
- ▶ all **global** PROMELA **variables** and **constants** of type **bool/bit** are  $\in For_{BTL}$
- ▶ if  $e_1$  and  $e_2$  are numerical PROMELA expressions, then all of  **$e_1 == e_2$ ,  $e_1 != e_2$ ,  $e_1 < e_2$ ,  $e_1 \leq e_2$ ,  $e_1 > e_2$ ,  $e_1 \geq e_2$**  are  $\in For_{BTL}$
- ▶ if  $P$  is a process and  $l$  is a label in  $P$ , then  **$P@l$**  is  $\in For_{BTL}$  ( $P@l$  reads “ $P$  is at  $l$ ”)
- ▶ if  $\phi$  and  $\psi$  are formulas  $\in For_{BTL}$ , then all of

! $\phi$ ,  $\phi \&\& \psi$ ,  $\phi \parallel \psi$ ,  $\phi \rightarrow \psi$ ,  $\phi \leftrightarrow \psi$   
 $[\ ]\phi$ ,  $\langle \rangle \phi$ ,  $\phi \cup \psi$

are  $\in For_{BTI}$

# Semantics of Boolean Temporal Logic

A trace  $\tau$  through a PROMELA model  $M$



- ▶  $\mathcal{I}_j$  maps each variable in  $M$  to its current value
- ▶  $\mathcal{L}_j$  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)

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

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

Evaluating other formulas  $\in For_{BTL}$  in traces  $\tau$ : see previous lecture

# 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)`

“**Throughout** a run, the value of `critical` is at most 1.”

or, equivalently:

“It will **never happen** that the value of `critical` is higher than 1.”

Any violating run would have `(critical > 1)` after *finite* time

# Applying Temporal Logic to Critical Section Problem

We want to **verify**  $\square(\text{critical} \leq 1)$  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 -DSAFETY -o pan pan.c
> ./pan -N name
```

Demo: target/safety1.pml

- The '`ltl name { TL-formula }`' construct must be part of your lab submission!

ltl definitions not part of Ben Ari's book ( $SPIN \leq 6$ ): ignore 5.3.2, etc.

# Model Checking a Safety Property using Web Interface

1. add definition of TL formula to PROMELA file

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

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

can define more than one formula

2. load PROMELA file into web interface
3. ensure **Safety** is selected
4. enter name of LTL formula in according field
5. select Verify

Demo: safety1.pml



# Model Checking a Safety Property using JSPIN

1. add definition of TL formula to PROMELA file

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

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

can define more than one formula

2. load PROMELA file into JSPIN
3. write *name* in 'LTL formula' field
4. ensure Safety is selected
5. select Verify
  - ▶ (corresponds to command line `./pan -N name ...`)
6. (if necessary) select Stop to terminate too long verification

Demo: `safety1.pml`

# Temporal Model Checking without Ghost Variables

We want to verify mutual exclusion **without using ghost variables**.

```
bool inCriticalP = false , inCriticalQ = false;
```

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

```
/* similar for process Q with same label cs: */
```

```
ltl mutualExcl { []!(P@cs && Q@cs) }
```

Demo: start/noGhost.pml

# Never Claims: Processes trying to show user wrong

## Büchi automaton, as PROMELA process, for negated property

1. Negated TL formula translated to 'never' process
2. Accepting locations in Büchi automaton represented with help of **accept** labels ("acceptxxx:")
3. If one of these reached infinitely often, the orig. property is violated

## Example (Never claim for $\langle \rangle p$ , simplified for readability)

```
never { /* ! $\langle \rangle p$  */  
  accept_xyz: /* passed  $\infty$  often iff ! $\langle \rangle p$  holds */  
  do  
    :: !p  
  od  
}
```

# Liveness Properties

## Liveness Properties

- ▶ state that something good ( $\phi$ ) **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**  $\langle \rangle \text{csp}$  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 using JSPIN

1. open PROMELA file `liveness1.pml`
2. write `ltl PwillEnterCS { <>csp }` in PROMELA file  
(as first `ltl` formula)
3. ensure that **Acceptance** is selected (for liveness properties)  
(SPIN will search for *accepting* cycles through the never claim)
4. *for the moment* uncheck Weak Fairness (see discussion below)
5. select Verify

# 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 using JSPIN

Always check **Weak fairness** when verifying liveness

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



# Model Checking Liveness using Web Interface

1. add definition of TL formula to PROMELA file

**Example** `ltl pWillEnterC { <>csp }`

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

can define more than one formula

2. load PROMELA file into web interface
3. ensure **Acceptance** is selected (for liveness properties)
4. enter name of LTL formula in according field
5. ensure **Weak fairness** is checked
6. select Verify

Demo: liveness1.pml

# 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** check acceptance cycles, **-f** weak fairness

Demo: `start/liveness1.pml`

# 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
- ▶ Prove termination
- ▶ Here, weak fairness is needed, *and sufficient*

Demo: `target/fair.pml`

# LTL Model Checking—Theory

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

Given:

- ▶ transition system  $\mathcal{T}$  (e.g., derived from a PROMELA program)
- ▶ LTL formula  $\phi$

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

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

# LTL Model Checking—Preview

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

1. Construct **generalised Büchi automaton**  $\mathcal{GB}_{\neg\phi}$  for **negation** of  $\phi$
2. Construct an equivalent **normal Büchi automaton**  $\mathcal{B}_{\neg\phi}$ , i.e.,

$$\mathcal{L}^\omega(\mathcal{B}_{\neg\phi}) = \mathcal{L}^\omega(\mathcal{GB}_{\neg\phi})$$

3. Construct **product**  $\mathcal{T} \otimes \mathcal{B}_{\neg\phi}$
4. Analyse whether  $\mathcal{T} \otimes \mathcal{B}_{\neg\phi}$  has a  
**path  $\pi$  looping through an 'accepting node'**
5. If such a  $\pi$  is found, then

$$\mathcal{T} \not\models \phi$$

and

$\sigma_\pi$  is a counter example.

If no such  $\pi$  is found, then

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

# When What?

## this lecture

- 3.–5. product of transition system and Büchi automaton (construction and analysis)

## next lecture

1. translating LTL into generalised Büchi automata
2. generalised Büchi automata and their normalisation

A model checking graph is a directed graph with initial and accepting nodes.

## Definition (Model Checking Graph)

A **model checking graph**  $(N, \rightarrow, N_0, N_a)$  is composed of:

- ▶ finite, non-empty set of **nodes**  $N$
- ▶ an 'arrow' relation  $\rightarrow \subseteq N \times N$
- ▶ a non-empty set of **initial** nodes  $N_0 \subseteq N$
- ▶ a set of **accepting** nodes  $N_a \subseteq N$



# Product of Transition System and Büchi Automaton

In the following, we assume without further mention:

1. transition systems **without terminal states**:

$$\{s' \in S \mid s \rightarrow s'\} \neq \emptyset \text{ for all states } s \in S$$

2. **total** Büchi automata:

$$\delta(q, a) \neq \{\} \text{ for all } q \in Q \text{ and } a \in \Sigma$$

Can always be achieved by adding 'trap states' or 'trap locations', resp.

# Product of Transition System and Büchi Automaton

We assume a set of atomic propositions  $AP$ .

## Definition (Product of Transition System and Büchi Automaton)

Let  $\mathcal{T} = (S, \rightarrow, S_0, L)$  be a transition system over  $AP$  and  $\mathcal{B} = (Q, \delta, Q_0, F)$  be a Büchi automaton over the alphabet  $2^{AP}$ . Then,  $\mathcal{T} \otimes \mathcal{B}$  is the following **model checking graph**:

$$\mathcal{T} \otimes \mathcal{B} = (S \times Q, \rightarrow', N_0, N_a)$$

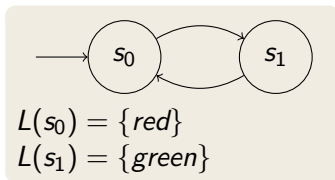
where:

- ▶  $\langle s, q \rangle \rightarrow' \langle s', q' \rangle$  iff  $s \rightarrow s'$  and  $(q, L(s'), q') \in \delta$
- ▶  $N_0 = \{ \langle s_0, q \rangle \mid s_0 \in S_0 \text{ and } \exists q_0 \in Q_0. (q_0, L(s_0), q) \in \delta \}$
- ▶  $N_a = \{ \langle s, q \rangle \mid q \in F \}$

# Model Checking Example

Assume  $AP = \{red, green\}$

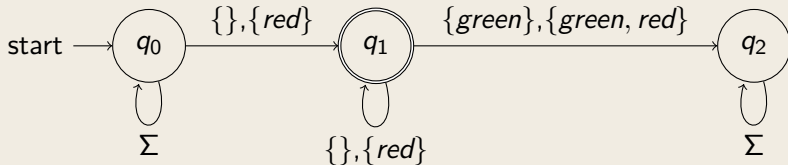
$\mathcal{T}$ :



We want to show “infinitely often *green*”:  $\phi \equiv \Box \Diamond green$

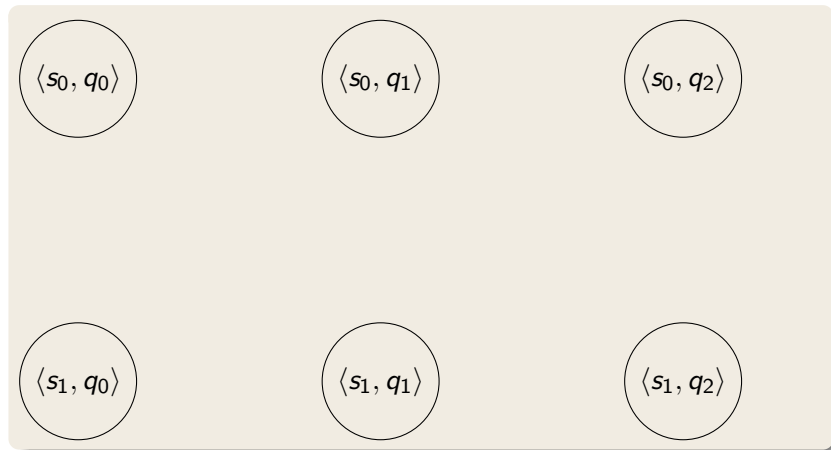
Construct BA  $\mathcal{B}_{\neg\phi}$  for negation:  $\neg\phi \equiv \neg\Box\Diamond green \equiv \Diamond\Box\neg green$

$\mathcal{B}_{\neg\phi}$ :



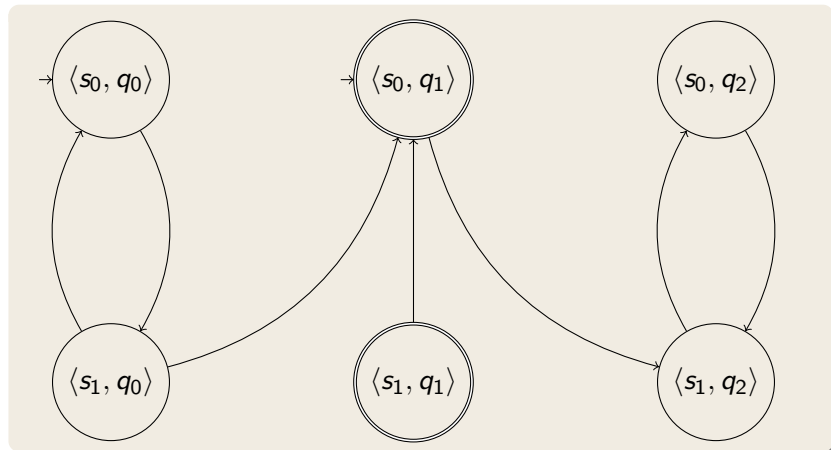
# Model Checking Example

Model checking graph  $\mathcal{T} \otimes \mathcal{B}_{\neg\phi}$ :



# Model Checking Example

Model checking graph  $\mathcal{T} \otimes \mathcal{B}_{\neg\phi}$ :



has no path looping through an accepting node!

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

**Ben-Ari** Chapter 5

**except** Sections 5.3.2, 5.3.3, 5.4.2

(`ltl` construct replaces `#define` and `-f` option of SPIN)