Protocol Verification

A Brief Introduction to Model Checking and Temporal Logic

Andrés Goens (U. of Amsterdam)

SPLV 2024 @ Strathclyde

29.07.2024 - 02.08.2024

Motivation

Protocol Verification?

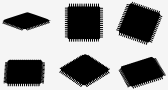
Protocols

Examples of protocols

- Distributed systems (e.g. paxos) 

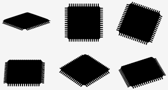
Protocols

Examples of protocols

- Distributed systems (e.g. paxos) 
- Hardware (e.g. cache coherence) 

Protocols

Examples of protocols

- Distributed systems (e.g. paxos) 
- Hardware (e.g. cache coherence) 
- Cryptographic protocols (e.g. TLS) 

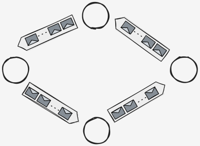
Verification

Examples of properties

- Fairness 

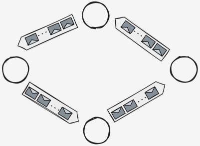
Verification

Examples of properties

- Fairness 
- Deadlock-freedom 

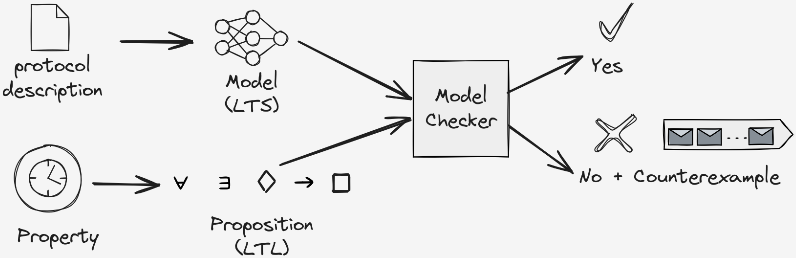
Verification

Examples of properties

- Fairness 
- Deadlock-freedom 
- Safety 

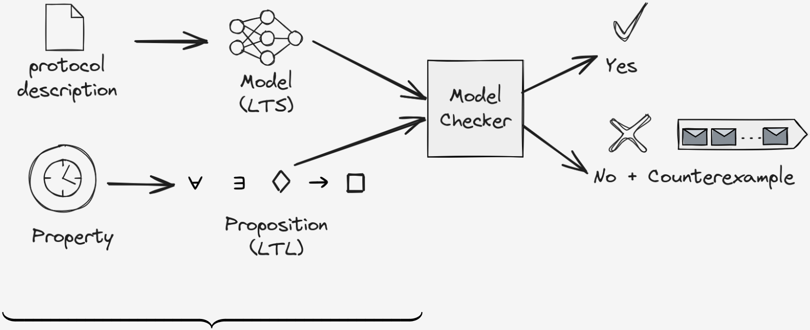
Protocol Verification

What this course is about



Protocol Verification

What this course is about



This is what we'll cover

Overview of the course

What you *will* (hopefully) know by the end

- Labeled transition systems (LTS)
- Modeling languages (promela)
- (Propositional) Linear Temporal Logic (LTL)
- Examples!

Overview of the course

What you *will* (hopefully) know by the end

- Labeled transition systems (LTS)
- Modeling languages (promela)
- (Propositional) Linear Temporal Logic (LTL)
- Examples!

What you will *not* (necessarily) know by the end

- Other logics (e.g. CTL*, μ calculus)
- How model checking works internally (decision procedures)

Modelling Protocols

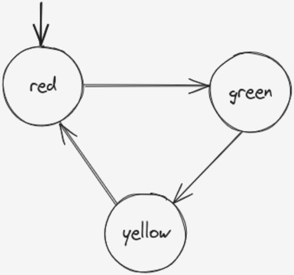
Labeled Transition Systems

Definition (Labeled Transition Systems)

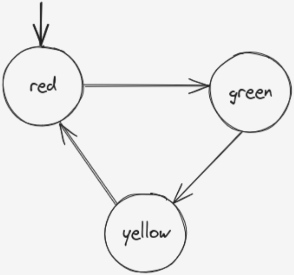
A labeled transition system is a tuple of the form $(S, \text{Act}, \rightarrow, S_0, \text{AP}, L)$, where S is a set of states, $S_0 \subseteq S$ a subset of initial states, Act is a set (of actions), $\rightarrow \subseteq \text{Act} \times S \times S$ is a (transition) relation, AP is a set (of atomic propositions) and $L : S \rightarrow \text{Pow}(\text{AP})$ is a (labeling) function.

Example: Traffic Light

Example: Traffic Light

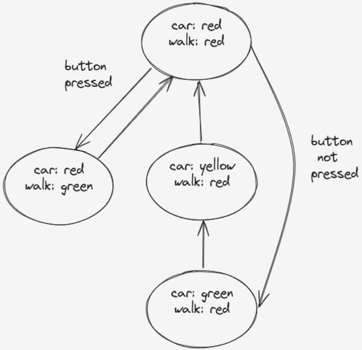


Example: Traffic Light

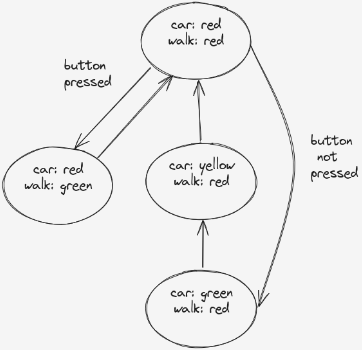


- $S = \{\text{red}, \text{green}, \text{yellow}\}, \; S_0 = \text{red}$
- $\text{Act} = \{*\}$
- $\rightarrow = \{(*, \text{red}, \text{green}), (*, \text{green}, \text{yellow}), (*, \text{yellow}, \text{red})\}$
- $\text{AP} = L = \emptyset$.

Two Traffic Lights



Two Traffic Lights



- $\text{Act} = \{\epsilon, \text{button pressed}, \text{no button pressed}\}$
- $\text{AP} = \{\text{Pedestrians can go}, \text{Cars can go}\}$
- $L = \text{cars: red, walk: green} \mapsto \{\text{Pedestrians can go}\}, \ldots$

Interleaving

Two traffic lights $\not\leftrightarrow$ One LTS

Interleaving

Two traffic lights $\nleftrightarrow$ One LTS

Definition (Interleaving)

Let $TS_i = (S_i, \mathrm{Act}_i, \rightarrow_i, S_{0,i}, \mathrm{AP}_i, L_i), i = 1, 2$ be two transition systems. We define the transition system $TS_1 \interleave TS_2 :=$ $(S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \rightarrow, S_{0,1} \times S_{0,2}, \mathrm{AP}_1 \cup \mathrm{AP}_2, L_1 \times L_2)$, where $L_1 \times L_2 : S_1 \times S_2 \rightarrow \mathrm{Pow}(\mathrm{AP}_1 \cup \mathrm{AP}_2)$ is defined as $(L_1 \times L_2)(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$ and $\rightarrow$ is defined by

$$\frac{s_1 \rightarrow_1^{\alpha} s_1'}{(s_1, s_2) \rightarrow^{\alpha} (s_1', s_2)} \qquad \frac{s_2 \rightarrow_2^{\alpha} s_2'}{(s_1, s_2) \rightarrow^{\alpha} (s_1, s_2')} \ .$$

We call this construction the *interleaving* of TS_1 and TS_2.

Interleaving

Two traffic lights $\nleftrightarrow$ One LTS

Definition (Interleaving)

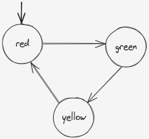
Let $TS_i = (S_i, \mathrm{Act}_i, \rightarrow_i, S_{0,i}, \mathrm{AP}_i, L_i), i = 1, 2$ be two transition systems. We define the transition system $TS_1 \,|||\, TS_2 :=$ $(S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \rightarrow, S_{0,1} \times S_{0,2}, \mathrm{AP}_1 \cup \mathrm{AP}_2, L_1 \times L_2)$, where $L_1 \times L_2 : S_1 \times S_2 \rightarrow \mathrm{Pow}(\mathrm{AP}_1 \cup \mathrm{AP}_2)$ is defined as $(L_1 \times L_2)(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$ and $\rightarrow$ is defined by

$$\frac{s_1 \rightarrow_1^\alpha s_1'}{(s_1, s_2) \rightarrow^\alpha (s_1', s_2)} \qquad \frac{s_2 \rightarrow_2^\alpha s_2'}{(s_1, s_2) \rightarrow^\alpha (s_1, s_2')} \ .$$

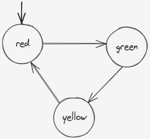
We call this construction the *interleaving* of TS_1 and TS_2.

Note that this means the two TS are *independent*

Example: Intearleaving

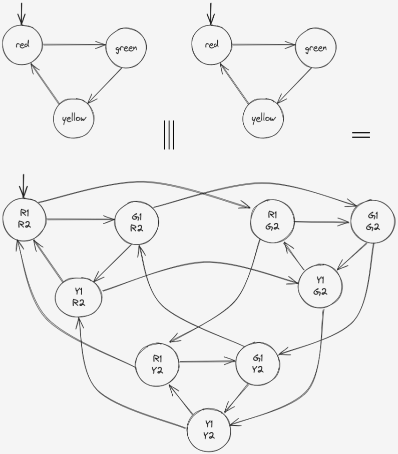


$|||$



$=$

Example: Intearleaving



Parallel Composition

Definition (Handshake)

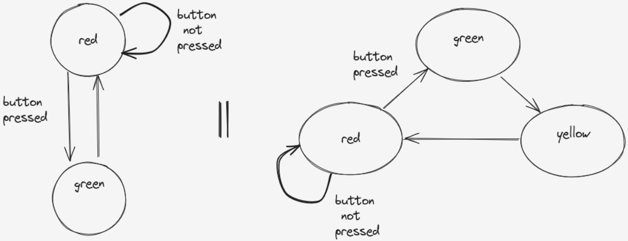
Let $TS_i = (S_i, \mathrm{Act}_i, \rightarrow_i, S_{0,i}, \mathrm{AP}_i, L_i), i = 1, 2$ be two transition systems and $H \subseteq \mathrm{Act}_1 \cap \mathrm{Act}_2$. We define the transition system $TS_1 \parallel_H TS_2 := (S_1 \times S_2, \mathrm{Act}_1 \cup \mathrm{Act}_2, \rightarrow$ $, S_{0,1} \times S_{0,2}, \mathrm{AP}_1 \cup \mathrm{AP}_2, L_1 \times L_2)$, where $\rightarrow$ is defined by:

$$\frac{s_1 \rightarrow_1^\alpha s_1' \quad \alpha \notin H}{(s_1, s_2) \rightarrow^\alpha (s_1', s_2)} \qquad \frac{s_2 \rightarrow_1^\alpha s_2' \quad \alpha \notin H}{(s_1, s_2) \rightarrow^\alpha (s_1, s_2')}$$

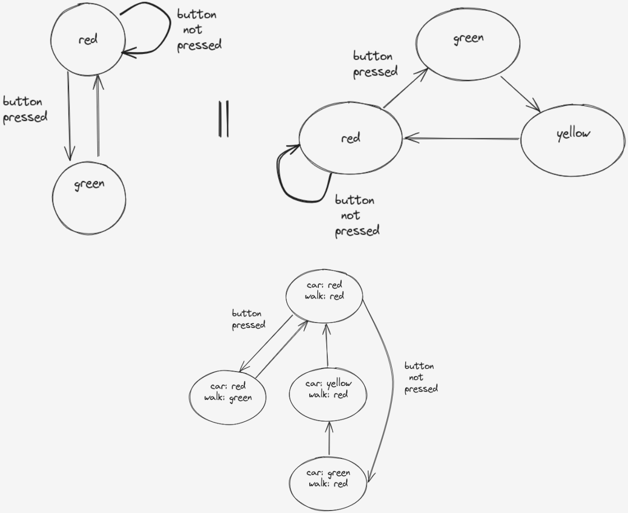
$$\frac{s_1 \rightarrow_1^\alpha s_1' \quad s_2 \rightarrow_2^\alpha s_2' \quad \alpha \in H}{(s_1, s_2) \rightarrow^\alpha (s_1', s_2')}$$

We call this the *parallel composition with handshake H*. When $H = \mathrm{Act}_1 \cap \mathrm{Act}_2$, we omit H.

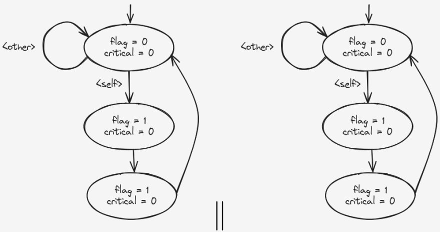
Two Traffic Lights, revisited



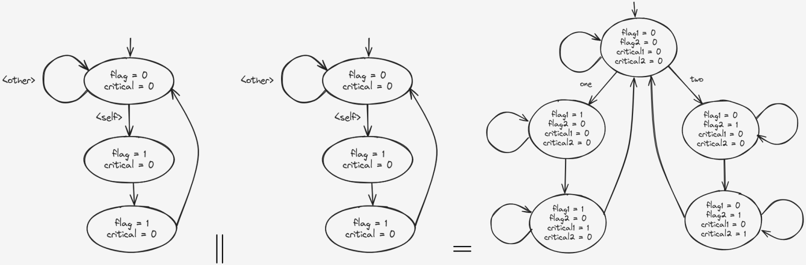
Two Traffic Lights, revisited



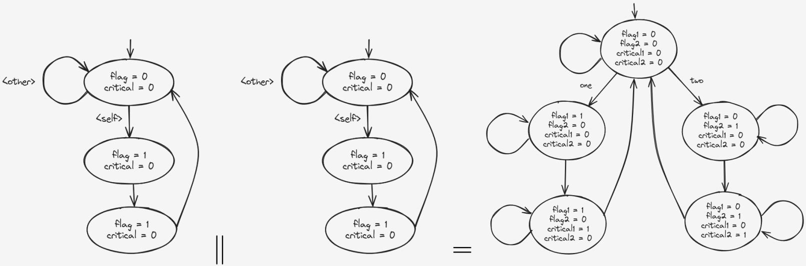
Concurrency: Message Passing



Concurrency: Message Passing

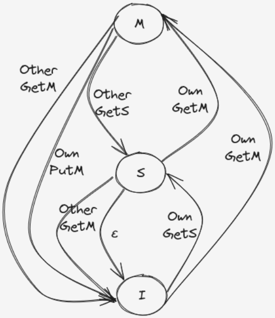


Concurrency: Message Passing

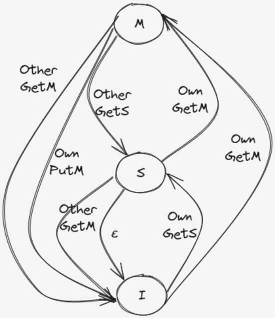


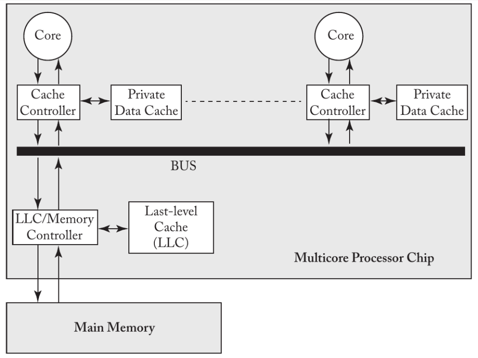
Assumption: atomicity of read-modify-writes here. Reasonable?

MSI Cache Coherency Protocol



MSI Cache Coherency Protocol





Source: Nagarajan, Vijay, et al. A primer on memory consistency and cache coherence. Springer Nature, 2020.

State Graph

- TS $\neq$ Graphs

State Graph

- TS $\neq$ Graphs
- Visualization (graphs): very useful!

State Graph

- TS $\neq$ Graphs
- Visualization (graphs): very useful!

Definition (Predecessors/Successors)

Let $TS = (S, \text{Act}, \rightarrow, S_0, \text{AP}, L)$ be a transition system. For $s \in S, \alpha \in \text{Act}$, we define $\text{Post}(s, \alpha) := \{s' \in S \mid s \rightarrow^{\alpha} s'\}, \text{Post}(s) := \bigcup_{\alpha \in \text{Act}} \text{Post}(s, \alpha)$ as the successors of s, and similarly Pre for the predecessors.

State Graph

- TS $\neq$ Graphs
- Visualization (graphs): very useful!

Definition (Predecessors/Successors)

Let $TS = (S, \text{Act}, \to, S_0, \text{AP}, L)$ be a transition system. For $s \in S, \alpha \in \text{Act}$, we define $\text{Post}(s,\alpha) := \{s' \in S \mid s \to^{\alpha} s'\}, \text{Post}(s) := \bigcup_{\alpha \in \text{Act}} \text{Post}(s,\alpha)$ as the successors of s, and similarly Pre for the predecessors.

Definition (State Graph)

Let $TS = (S, \text{Act}, \to, S_0, \text{AP}, L)$ be a transition system. We call the directed graph $G(TS) = (S, E)$ the state graph of TS, where $E = \{s, s' \in S \times S \mid s \in S, s' \in \text{Post}(s)\}$

Path Fragments

Definition (Path fragments)

Let $TS = (S, \text{Act}, \rightarrow, S_0, \text{AP}, L)$ be a transition system. A sequence $\pi = \pi_0\pi_1\pi_2 \ldots \in (S)_{\mathbb{N}}$ is called a *path fragment* if $\pi_{i+1} \in \text{Post}(\pi_i) \forall i \in \mathbb{N}$. It is called *finite* if it is a finite sequence $(\pi_i)_{i=0}^{N}$instead.

For a path fragment π, we denote the i-th element by $\pi[i]$ and similarly the sub-sequence $(\pi_k)_{k=i}^{j}$ by $\pi[i..j]$

Path Fragments

Definition (Path fragments)

Let $TS = (S, \text{Act}, \rightarrow, S_0, \text{AP}, L)$ be a transition system. A sequence $\pi = \pi_0\pi_1\pi_2 \ldots \in (S)_{\mathbb{N}}$ is called a *path fragment* if $\pi_{i+1} \in \text{Post}(\pi_i) \forall i \in \mathbb{N}$. It is called *finite* if it is a finite sequence $(\pi_i)_{i=0}^{N}$instead.
For a path fragment π, we denote the i-th element by $\pi[i]$ and similarly the sub-sequence $(\pi_k)_{k=i}^{j}$ by $\pi[i..j]$

Sequences of transitions = path framgents through the state graph

Paths

Definition (Initial path fragment)

A path fragment π is called *initial*, if it starts at an initial statei, i.e. $\pi_0 \in S_0$.

Paths

Definition (Initial path fragment)

A path fragment π is called *initial*, if it starts at an initial statei, i.e. $\pi_0 \in S_0$.

Definition (Maximal path fragment)

A path fragment π is called a maximal, if it is not a proper prefix $\pi \sqsubsetneq \pi'$ of another path fragment π', i.e. it cannot be extended.

Paths

Definition (Initial path fragment)

A path fragment π is called *initial*, if it starts at an initial statei, i.e. $\pi_0 \in S_0$.

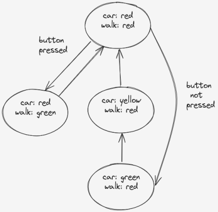
Definition (Maximal path fragment)

A path fragment π is called a maximal, if it is not a proper prefix $\pi \subsetneq \pi'$ of another path fragment π', i.e. it cannot be extended.

Definition (Path)

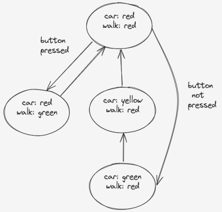
A path fragment π is called a *path* if it is initial and maximal.

Example: Paths in Traffic Light



A Typical Traffic Light in the UK?

Example: Paths in Traffic Light



A Typical Traffic Light in the UK?

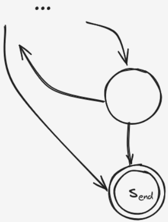
Non-example

Finite vs Infinite Paths

finite path fragments can be extended to infinite ones, but...

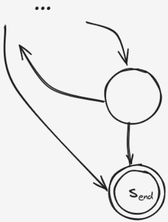
Finite vs Infinite Paths

finite path fragments can be extended to infinite ones, but...



Finite vs Infinite Paths

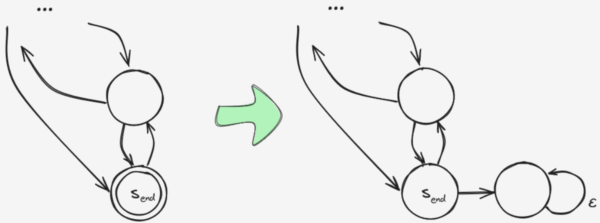
finite path fragments can be extended to infinite ones, but...



$$\text{Post}(s) = \varnothing$$

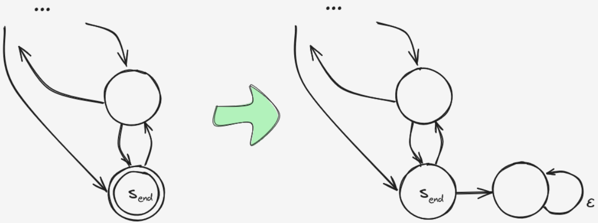
End States

Modeling end states with infinite paths



End States

Modeling end states with infinite paths



Assumption

For the rest of this course we assume no end states s with $\text{Post}(s) = \emptyset$.

Traces

- Paths $\triangleq$ sequences of states $\in S$

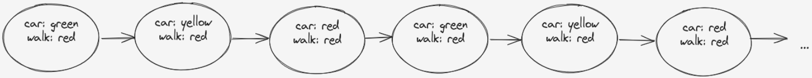
Traces

- Paths $\triangleq$ sequences of states $\in S$
- Properties defined over AP, not S

Definition (Traces)

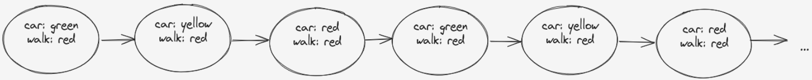
Let π be a path fragment. We define the *trace* of π as the sequence $L(\pi) \in (\mathbb{N} \to \mathrm{Pow}(\mathrm{AP}))$ as the sequence given by $(L(\pi))_i = L(\pi_i) \forall i \in \mathbb{N}$, and similarly for a finite path fragment. For $s \in S$ we define $\mathrm{Traces}(s)$ as the set of traces for path fragments starting at s, and $\mathrm{Traces}(TS) = \bigcup_{s \in S_0} \mathrm{Traces}(s)$.

Example: Traces

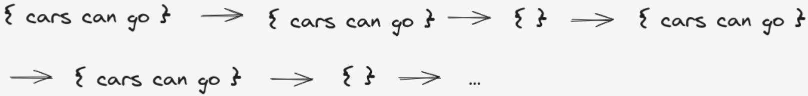


Corresponds to

Example: Traces



Corresponds to



Equivalence of LTSs

- Many notions of equivalence.

Equivalence of LTSs

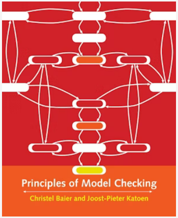
- Many notions of equivalence.
- Today: one

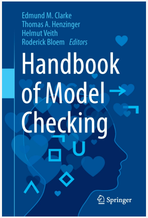
Definition (Trace Equivalence)

Let $TS_i, i = 1, 2$ be two tranisition systems with $\mathrm{AP}_1 = \mathrm{AP}_2$. We say TS_1 and TS_2 are *trace equivalent* if $\mathrm{Traces}(TS_1) = \mathrm{Traces}(TS_2)$.

References

Main references for this course:



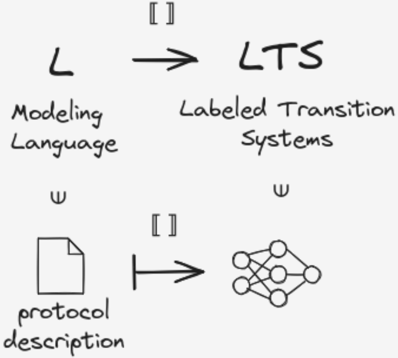


- Baier, Christel, and Joost-Pieter Katoen. Principles of model checking. MIT press, 2008.
- Clarke, Edmund M., et al., eds. Handbook of model checking. Vol. 10. Cham: Springer, 2018.

Modeling Languages: An Introduction to Promela

Modelling Languages

Core Idea



Promela

- Spin: mature model checker (>30 years of development)

Promela

- Spin: mature model checker ($>$30 years of development)
- Promela = **Pro**tocol/cess **me**ta **la**nguage

Promela

- Spin: mature model checker (>30 years of development)
- Promela = **Pro**tocol/cess **me**ta **la**nguage
- C-inspired syntax

Hello Promela

```
init{
    int num = 11 * 23 * 8;
    printf("Hello SPLV %d\n", num);
}
```

Hello Promela

```
init{
    int num = 11 * 23 * 8;
    printf("Hello SPLV %d\n", num);
}
```

```
→  splv24 git:(master) ✗ spin promela-examples/hello.pml
      Hello SPLV 2024
1 process created
```

Do Blocks

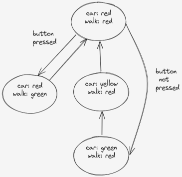
```
#define N 100

proctype counter(int i){
    do // repeats indefinitely
    :: (i < N) -> i = i + 1  // guarded increase
    :: (i >= N) -> break // break do loop
    od
    end: skip // declare a (valid) end state
}

init{
    run counter(0)
}
```

Promela: Traffic Lights

```
mtype = {red, green, yellow}
mtype car = red;
mtype walk = red;

active proctype TrafficLight(){
    do
    :: (walk == red && car == red) -> car = green
    :: (walk == red && car == red) -> walk = green
    :: (car == red && walk == green) -> walk = red
    :: car == green -> car = yellow
    :: car == yellow -> car = red
    od
}
```

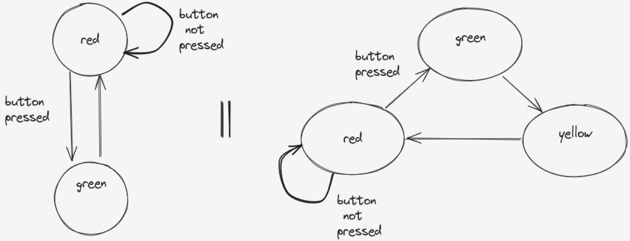
Promela: Traffic Lights

```
mtype = {red, green, yellow}
mtype car = red;
mtype walk = red;

active proctype TrafficLight(){
    do
    :: (walk == red && car == red) -> car = green
    :: (walk == red && car == red) -> walk = green
    :: (car == red && walk == green) -> walk = red
    :: car == green -> car = yellow
    :: car == yellow -> car = red
    od
}
```



Composition

Recall:



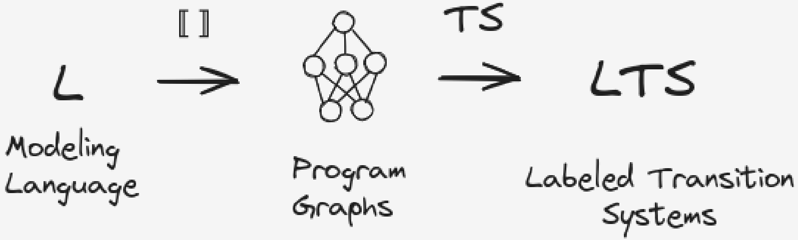
Communication (Channels)

```
mtype = {red, green, yellow}
mtype car = red;
mtype walk = red;

// Channel of size 0 = synchronous communication
chan press = [0] of {bool};

active proctype PedestrianButton(){
    do
    :: press!true // send `true`
    :: press!false // send `false`
    od
}
```

Communication (Channels) contd.

```
active proctype TrafficLight(){
    bool button_pressed = false;
    do
    :: (walk == red && car == red) ->
       press?button_pressed; //receive pressed
       if
       :: button_pressed -> walk = green
       :: !button_pressed -> car = green
       fi
    :: (car == red && walk == green) -> walk = red
    :: car == green -> car = yellow
    :: car == yellow -> car = red
    od
}
```

Program Graphs



Program Graphs (ctd.)

Core ideas:

- States = Program locations (Loc) $\times$ values of variables $[\![\Gamma]\!]$

Program Graphs (ctd.)

Core ideas:

- States = Program locations (Loc) $\times$ values of variables $[\![\Gamma]\!]$
- Conditions over variables in context Γ: $\mathrm{Cond}(\Gamma)$ (propositional logic)

Program Graphs (ctd.)

Core ideas:

- States = Program locations (Loc) $\times$ values of variables $[\![\Gamma]\!]$
- Conditions over variables in context Γ: $\mathrm{Cond}(\Gamma)$ (propositional logic)
- conditional transition relation:

$$\hookrightarrow \subseteq \mathrm{Cond}(\Gamma) \times \mathrm{Act} \times \mathrm{Loc} \times \mathrm{Loc}$$

Program Graphs (ctd.)

Core ideas:

- States = Program locations (Loc) $\times$ values of variables $[\![\Gamma]\!]$
- Conditions over variables in context Γ: $\mathsf{Cond}(\Gamma)$ (propositional logic)
- conditional transition relation:

$$\hookrightarrow \subseteq \mathsf{Cond}(\Gamma) \times \mathsf{Act} \times \mathsf{Loc} \times \mathsf{Loc}$$

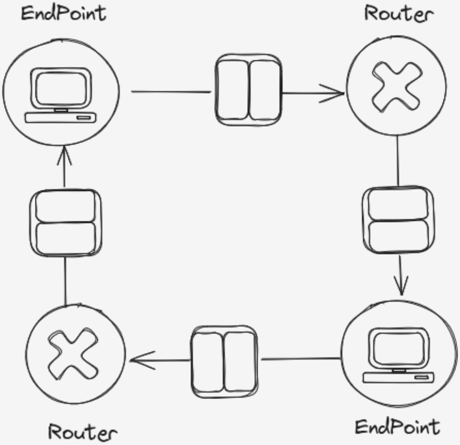
- Transition relation from this:

$$\frac{l \hookrightarrow^{g,\alpha} l' \quad \eta \models g}{(l,\eta) \rightarrow^{\alpha} (l', ([\![\alpha]\!])(\eta))}$$

Example: MP Concurrency

```
bool flag[2]; //flag for entering critical section
byte num_crit; //how many processes in critical section

active [2] proctype user()        // two processes
{
do
::
       flag[_pid] = 1;
       flag[1 - _pid] == 0 ->
           num_crit = num_crit + 1; // enter
           num_crit = num_crit - 1; // exit
       flag[_pid] = 0;
od
}
```

Example: Buffers



Example: Buffers in Promela

```
mtype = {request, response, nil}

proctype Router(chan buffer_from, buffer_to){
    mtype msg = nil;
    do /* a router just keeps forwarding messages */
    :: buffer_from?msg  -> buffer_to!msg
    od
}
```

Example: Buffers in Promela (ctd.)

```
proctype EndPoint(chan buffer_from, buffer_to){
    mtype msg = nil;
    do
    :: atomic{ (msg == nil) && buffer_from?[msg]
    -> buffer_from?msg}
    :: atomic{ (msg == request)
    -> buffer_to!response; msg = nil }
    :: atomic{ (msg == response) ->  msg = nil }
    :: buffer_to!request
    od
}
```

Semantics of channels

Core idea:

Semantics of channels

Core idea:

- Extend actions Act with set of communication actions Comm

Semantics of channels

Core idea:

- Extend actions Act with set of communication actions Comm
- Comm: Actions $c!v$ and $c?x$ to send value v on channel c and receive into variable x.

Semantics of channels

Core idea:

- Extend actions Act with set of communication actions Comm
- Comm: Actions $c!v$ and $c?x$ to send value v on channel c and receive into variable x.
- Multiple program graphs: composition ($\|$) with matching actions built from $c?v/c!v$ pairs.

Modelling Properties

Models

What is a model?

Models

What is a model?

Model Theory 101

- Structures, e.g. groups, rings, fields, *labeled transition systems*

Model Theory 101

- Structures, e.g. groups, rings, fields, *labeled transition systems*
- Formulas in a given logic, e.g. $a = b$, $\exists c, a * c = 1$, $\Box(\neg p)$

Model Theory 101

- Structures, e.g. groups, rings, fields, *labeled transition systems*
- Formulas in a given logic, e.g. $a = b$, $\exists c, a * c = 1$, $\Box(\neg p)$
- Models $A \models \phi$, i.e. the formula ϕ holds in the structure A

(Modal) Logic

- Propositional logic $(P, Q, \ldots, \vee, \wedge, \neg)$

(Modal) Logic

- Propositional logic $(P, Q, \ldots, \vee, \wedge, \neg)$
- First-order logic $(P, Q, \ldots, \vee, \wedge, \neg, \exists, \forall)$

(Modal) Logic

- Propositional logic $(P, Q, \ldots, \vee, \wedge, \neg)$
- First-order logic $(P, Q, \ldots, \vee, \wedge, \neg, \exists, \forall)$
- Modal logic $(\ldots, \Box, \Diamond)$
  - $\Box \approx$ necessity
  - $\Diamond \approx$ possibility

Linear-Time Properties

Definition (LT Property)

Let $TS = (S, \text{Act}, \rightarrow, S_0, \text{AP}, L)$ be a transition system. A *linear property* of TS is a set of traces, i.e. sequences $P \subseteq \text{AP}^{\mathbb{N}}$ over atocmic propositions AP.

Linear-Time Properties

Definition (LT Property)

Let $TS = (S, \mathrm{Act}, \rightarrow, S_0, \mathrm{AP}, L)$ be a transition system. A *linear property* of TS is a set of traces, i.e. sequences $P \subseteq \mathrm{AP}^{\mathbb{N}}$ over atocmic propositions AP.

Idea: these are the admissible traces in TS

Linear-Time Properties

Definition (LT Property)

Let $TS = (S, \mathrm{Act}, \rightarrow, S_0, \mathrm{AP}, L)$ be a transition system. A *linear property* of TS is a set of traces, i.e. sequences $P \subseteq \mathrm{AP}^{\mathbb{N}}$ over atocmic propositions AP.

Idea: these are the admissible traces in TS

Definition (Satisfying an LT Property)

Let $TS = (S, \mathrm{Act}, \rightarrow, S_0, \mathrm{AP}, L)$ be a transition system and let P be a linear time property. We say that TS satisfies P, in symbols, $TS \models P$, iff $\mathrm{Traces}(TS) \subseteq P$.

Linear Temporal Logic (intro)

- Propositional logic + modal operators
  $(P, Q, \ldots, \vee, \wedge, \neg, \bigcirc, \cup, \Box, \Diamond)$

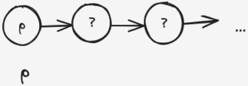
Linear Temporal Logic (intro)

- Propositional logic + modal operators $(P, Q, \ldots, \vee, \wedge, \neg, \bigcirc, \cup, \Box, \Diamond)$
  - $\Box \triangleq$ "Always"
  - $\Diamond \triangleq$ "Eventually"
  - $\bigcirc \triangleq$ "Next"
  - $\cup \triangleq$ "Until"

Linear Temporal Logic (intro)

- Propositional logic + modal operators ($P, Q, \ldots, \vee, \wedge, \neg, \bigcirc, \cup, \Box, \Diamond$)
  - $\Box \triangleq$ "Always"
  - $\Diamond \triangleq$ "Eventually"
  - $\bigcirc \triangleq$ "Next"
  - $\cup \triangleq$ "Until"
- Note: propositional logic (and LTL) has no quantifiers $\forall, \exists$ (!)

Intuition of LTL Operators



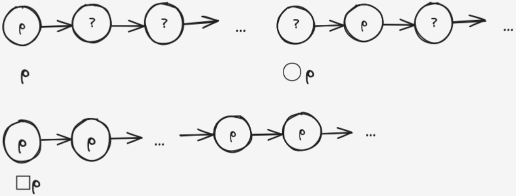
- $p \in AP$
- $\bigcirc \triangleq$ "Next"
- $\square \triangleq$ "Always"
- $\lozenge \triangleq$ "Eventually"
- $\cup \triangleq$ "Until"

Intuition of LTL Operators

- $p \in AP$
- $\bigcirc \triangleq$ "Next"
- $\Box \triangleq$ "Always"
- $\Diamond \triangleq$ "Eventually"
- $\cup \triangleq$ "Until"

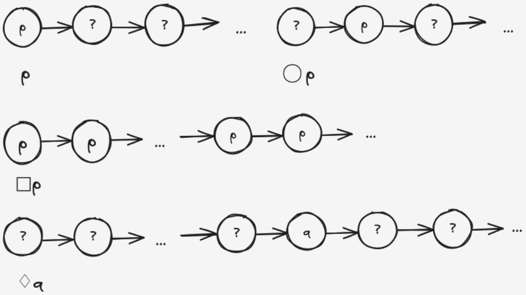
Intuition of LTL Operators

- $p \in AP$
- $\bigcirc \triangleq$ “Next”
- $\square \triangleq$ “Always”
- $\Diamond \triangleq$ “Eventually”
- $\cup \triangleq$ “Until”



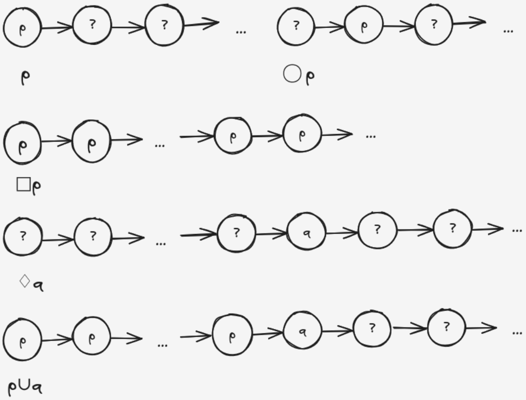
Intuition of LTL Operators

- $p \in AP$
- $\bigcirc \triangleq$ "Next"
- $\square \triangleq$ "Always"
- $\lozenge \triangleq$ "Eventually"
- $\cup \triangleq$ "Until"

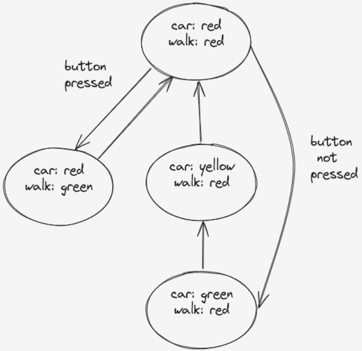


Intuition of LTL Operators

- $p \in AP$
- $\bigcirc \triangleq$ “Next”
- $\square \triangleq$ “Always”
- $\Diamond \triangleq$ “Eventually”
- $\cup \triangleq$ “Until”

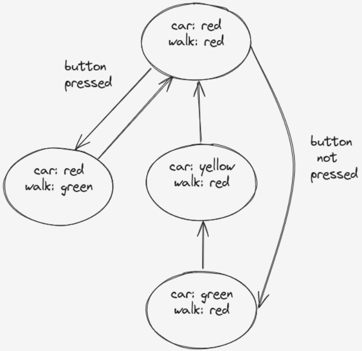


Example: Safety



“Cars and Pederstrians can never go at the same time ”

Example: Safety



"Cars and Pederstrians can never go at the same time " $\triangleq$
$\Box\neg(\text{cars can go} \wedge \text{pedestrians can go})$

LTL, Formally (Syntax)

Definition (Syntax of LTL)

Let AP be a set (of atomic propositions). Then, an LTL formula over AP is a word in the language defined by the grammar:

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \cup \varphi_2$$

We call the set of such formulae LTL_{AP}. When AP is clear from context, we also say φ is an LTL formula (and omit AP).

LTL, Formally (Semantics, I)

Definition (The "Models" Relation)

We define $\models$ as the minimial relation over traces and LTL formulae $\models \subseteq (\mathbb{N} \to \mathsf{Pow}(\mathsf{AP})) \times \mathsf{LTL}_{\mathsf{AP}}$, such that:

$$
\begin{array}{ll}
A \models \mathsf{true} & \\
A \models a \in \mathsf{AP} & \text{iff } a \in A_0 \\
A \models \varphi_1 \wedge \varphi_2 & \text{iff } A \models \varphi_1 \text{ and } A \models \varphi_2 \\
A \models \neg\varphi & \text{iff } A \not\models \varphi \\
A \models \bigcirc\varphi & \text{iff } A[1\ldots] = A_1 A_2 \ldots \models \varphi \\
A \models \varphi_1 \cup \varphi_2 & \text{iff } \exists j,\ A[j\ldots] \models \varphi_2 \text{ and } \forall i < j,\ \sigma[i\ldots] \models \varphi_1
\end{array}
$$

LTL, Formally (Semantics, II)

Definition (Semantics of LTL)

Let φ be an LTL formula over AP. We define
$\mathsf{Words}(\varphi) := \{\pi \in \mathsf{Pow}(\mathsf{AP})^{\mathbb{N}} \mid \pi \models \varphi\}$.

LTL, Formally (Semantics, II)

Definition (Semantics of LTL)

Let φ be an LTL formula over AP. We define
$\text{Words}(\varphi) := \{\pi \in \text{Pow}(\text{AP})^{\mathbb{N}} \mid \pi \models \varphi\}$.

Definition

We say the transition system *TS satisfies* φ (in symbols, $TS \models \varphi$), if $\text{Traces}(TS) \subseteq \text{Words}(\varphi)$.

Temporal Modalities

Definition ($\Diamond$ Operator)

For an LTL formula φ, we define the operator $\Diamond$ as

$$\Diamond\varphi := \mathsf{true} \cup \varphi$$

Temporal Modalities

Definition ($\Diamond$ Operator)

For an LTL formula φ, we define the operator $\Diamond$ as

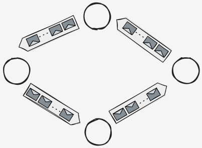
$$\Diamond\varphi := \mathsf{true} \cup \varphi$$

Definition ($\Box$ Operator)

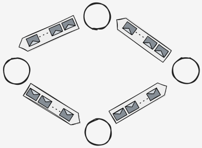
For an LTL formula φ, we define the operator $\Box$ as

$$\Box\varphi := \neg\Diamond\neg\varphi$$

Deadlocks

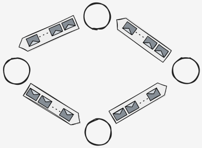
Recall: 

Deadlocks

Recall: 

Temporal logic?

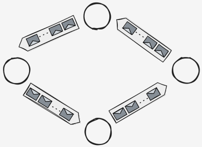
Deadlocks

Recall: 

Temporal logic?

Recall: we assumed no finite states

Deadlocks

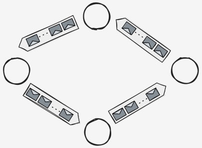
Recall: 

Temporal logic?

Recall: we assumed no finite states

- transformation is a deadlock check

Deadlocks

Recall: 

Temporal logic?

Recall: we assumed no finite states

- transformation is a deadlock check
- no deadlock $\triangleq \Box\neg$capture-state

Invariants

Invariant (property does not change) $\triangleq \Box P$

Examples:

- mutual exclusion: never two process in critical section
  $\Box(\mathit{crit} < 2)$
- cars and pedestrians don't go at the same time
  $\Box(\neg \text{cars can go} \lor \neg \text{pederstrians can go})$

Safety

Other safety properties: bad prefix

- Yellow should warn of red coming:
  $\Box(\neg(\text{yellow} \vee \text{red}) \rightarrow \bigcirc \neg \text{red})$

Definition

An LT property P over AP is called a *safety property*, if for all traces $\pi \in \text{Pow}\,(\text{AP})^{\mathbb{N}}$ there exists a finite prefix $\hat{\pi} \sqsubset \pi$ such that extensions of that prefix are disjoint from P, i.e.
$\{\pi' \in \text{Pow}\,(\text{AP})^{\mathbb{N}} \mid \hat{\pi} \sqsubset \pi'\} \cap P = \varnothing$

Fairness

"Everybody gets their turn"

- Unconditional $\Box\Diamond P$ ("Everybody gets their turn infinitely often")
- Strong $\Box\Diamond P \rightarrow \Box\Diamond Q$ ("Everybody who asks infinitely often, goes infinitely often")
- Weak $\Diamond\Box P \rightarrow \Box\Diamond Q$("Everybody who is waiting from some point on, gets their turn infinitely often")
- Fairness $\triangleq$ Unconditional $\wedge$ Strong $\wedge$ Weak

Fairness

"Everybody gets their turn"

- Unconditional $\Box\Diamond P$ ("Everybody gets their turn infinitely often")
- Strong $\Box\Diamond P \rightarrow \Box\Diamond Q$ ("Everybody who asks infinitely often, goes infinitely often")
- Weak $\Diamond\Box P \rightarrow \Box\Diamond Q$("Everybody who is waiting from some point on, gets their turn infinitely often")
- Fairness $\triangleq$ Unconditional $\wedge$ Strong $\wedge$ Weak

(Nondeterminism): Condition or constraint?

Liveness Properties

More generally, liveness are things of the type "good thing happen infinitely often"

- Traffic light's let people through: $\Box\Diamond\text{green}$
- Mutex lets processes do their work: $\Box((\Diamond\text{crit}_1) \wedge (\Diamond\text{crit}_2))$

Definition (Liveness — Alpern and Schneider)

An LT property P over AP is called a *liveness property*, if every finite word can be extended to a trace in the property P, i.e. for all $\hat{\pi} \in \text{Pow}(\text{AP})^*$ there exsits a $\pi \in P$ such that $\hat{\pi} \sqsubset \pi$.

Decomposition

- Safety properties: constrain finite behavior
- Liveness properties: constrain infinite behavior

Theorem

Let P be a linear time property P over AP, i.e. $P \subseteq \mathrm{Pow}\,(AP)^{\mathbb{N}}$. Then there exist a liveness property P_{live} and a safety property P_{safe} over AP, such that $P = P_{live} \cap P_{safe}$.

Decomposition Theorem

Proof.

(Sketch) The metric

$$d : \mathsf{Pow}(AP)^{\mathbb{N}} \times \mathsf{Pow}(AP)^{\mathbb{N}} \to \mathbb{R}_{\geqslant 0},$$
$$(\pi, \sigma) \mapsto \begin{cases} 0, & \text{if } \sigma = \pi \\ \frac{1}{|gcp(\sigma,\pi)|}, & \text{otherwise} \end{cases},$$

where $gcp(\sigma, \pi)$ denotes the greatest common prefix of σ and π, makes $\mathsf{Pow}(AP)^{\mathbb{N}}$ a metric space. Safety properties are the closed sets of the induced topology. We have

$$P = \underbrace{\bar{P}}_{:=P_{\mathsf{safe}}} \cap \underbrace{P \cup (\mathsf{Pow}(AP)^{\mathbb{N}}) \setminus \bar{P})}_{:=P_{\mathsf{live}}}$$

□

Model Checking with Spin

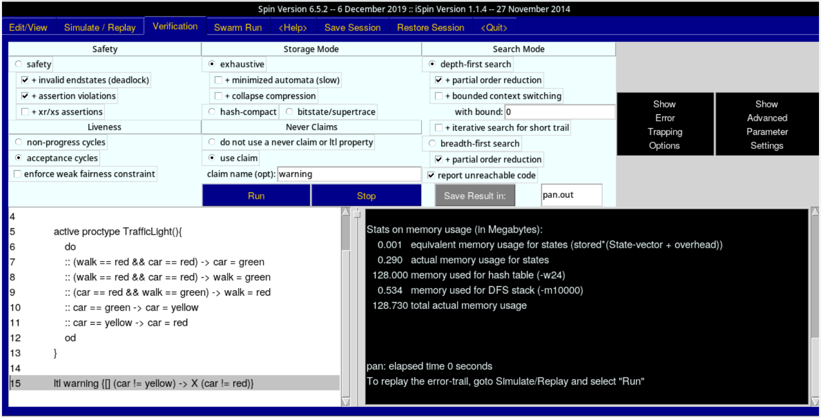
- Deadlocks: nothing additional! (`end` label)

Model Checking with Spin

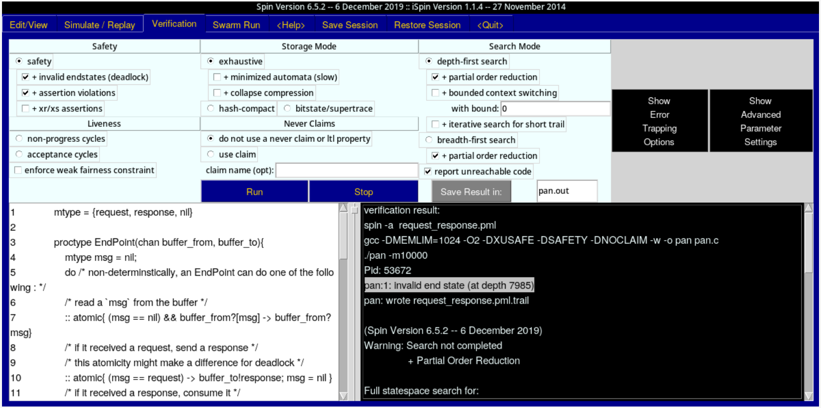
- Deadlocks: nothing additional! (`end` label)
- LTL Formulae: `never` claims

```
-f LTL Translate the LTL formula LTL into a never claim.
       This option reads a formula in LTL syntax from the second argument and  translates
       it into Promela syntax (a never claim, which is Promela's equivalent of a Bchi Au-
       tomaton).  The LTL operators are written: []  (always),  <>  (eventually),  and  U
       (strong  until).   There is no X (next) operator, to secure compatibility with the
       partial order reduction rules that are applied during  the  verification  process.
       If  the  formula contains spaces, it should be quoted to form a single argument to
       the SPIN command.
       This option has largely been replaced with the support for inline specification of
       ltl formula, in Spin version 6.0.
```

Example: Safety in Traffic Light



Example: Deadlock in Request-



A Word on Complexity

- Invariant checking (BFS) is linear in state space, formula, transitions (still large spaces!).
- General LTL model checking is PSPACE hard

A Word on Complexity

- Invariant checking (BFS) is linear in state space, formula, transitions (still large spaces!).
- General LTL model checking is PSPACE hard
- Mitigations:
  - Partial order reduction
  - Symmetry reduction
  - Abstraction (gradual refinements)
  - Symbolic model checking
  - ...