Hoare logic and Model checking

Part II: Model checking

Lecture 7: Introduction to model checking

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2020/21

Acknowledgements

These slides are heavily based on previous versions by Mike Gordon, Dominic Mulligan, Alan Mycroft, and Conrad Watt.

They are also inspired by slides and lecture notes by John Gallagher, Gourinath Banda, and Pierre Ganty, by Paul Gastin, by Orna Grumberg, by Arie Gurfinkel, by Daniel Kroening, by Antoine Miné, by Julien Schmaltz, by David A. Schmidt, and by Carsten Sinz and Tomáš Balyo.

Thanks to Neel Krishnaswami, David Kaloper Meršinjak, Jack Parkinson, Peter Rugg, Ben Simner, and two anonymous students, for remarks and reporting mistakes.

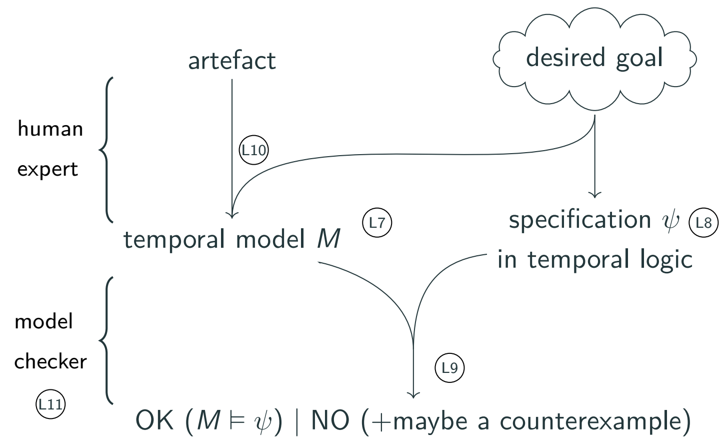
Warning about terminology and notation

The field of model checking is a terminological and notational nightmare.

In violation of the Countryside Code, we leave the field in a state worse than the one we found it in.

We also write meta-level and object-level constructors with the same symbols when not ambiguous.

Model checking


artefact
desired goal
human expert
L10
temporal model M
L7
specification ψ in temporal logic
L8
model checker
L11
L9
OK ($M \vDash \psi$) | NO (+maybe a counterexample)

This diagram gives a very static, top-down picture, but it is the feedback that provides the value.

Example

Suppose we are given an algorithm that is supposed to transfer, from one bank of the Cam to the other, using only a punt with seat for one, a wolf, a goat, and a cabbage[1].

The success criteria are

- safety: the cabbage and the goat, and the wolf and the goat, cannot be left alone on a bank;
- liveness: all three items are moved to the other bank.

[1]it is a large cabbage, so it takes up the whole seat

Example

How to model the problem?

- Option 1:

$$\mathcal{L} = -\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + i\bar{\psi}\not{D}\psi + \text{h.c.} + \bar{\psi}_i y_{ij}\psi_j\phi + \text{h.c.} + |D_\mu\phi|^2 - V(\phi) + ???$$

- Option 2: (G. Doré, anonymous (Wellcome coll.), G. Waddington)

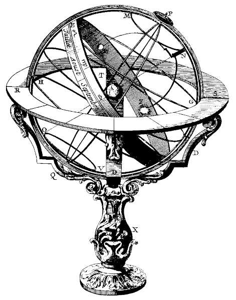
- Option 3: (apologies to the Phaistos cat)

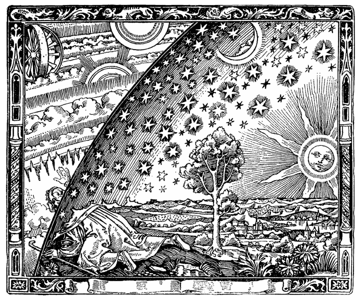
Side ::= 𐇽𐇼 | 𐇼𐇽 Item ::= 𐇵 | 𐇭 | 𐇬 | 𐇨

$$\text{State} \overset{def}{=} \text{Item} \to \text{Side}$$

...

How to find good models


A pretty good model of the solar system
Encyclopédie, Diderot, d'Alembert, *et al.*


The need to go beyond excessively simple models
Flammarion engraving, anonymous

"All models are wrong, some are useful" applies. The designer must ensure the model captures the significant aspects of the real system. Achieving it is a special skill, the acquisition of which requires thoughtful practice
— How Amazon Web Services Uses Formal Methods

Temporal models

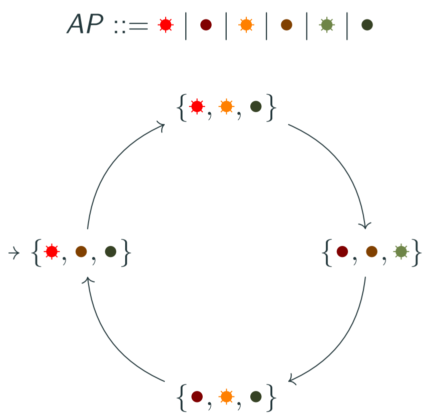
A **temporal model** over **atomic propositions** AP is a left-total transition system where states are labelled with some of AP, and where some states are distinguished as initial:

$$\begin{array}{ll}
M, \ldots \in \text{TModel} \overset{def}{=} & \\
\quad (S \in \text{Set}) \times & \text{states} \\
\quad\quad (S_0 \in \text{sub } S) \times & \text{initial states} \\
\quad\quad\quad (\textcircled{1}\ T\ \textcircled{2} \in \text{relation } S\ S) \times & \text{transition} \\
\quad\quad\quad\quad (\ell \in S \to \text{sub } AP) \times & \text{state labelling} \\
\quad\quad\quad\quad\quad (\forall s \in S.\, \exists s' \in S.\, s\ T\ s') & \text{left-total}
\end{array}$$

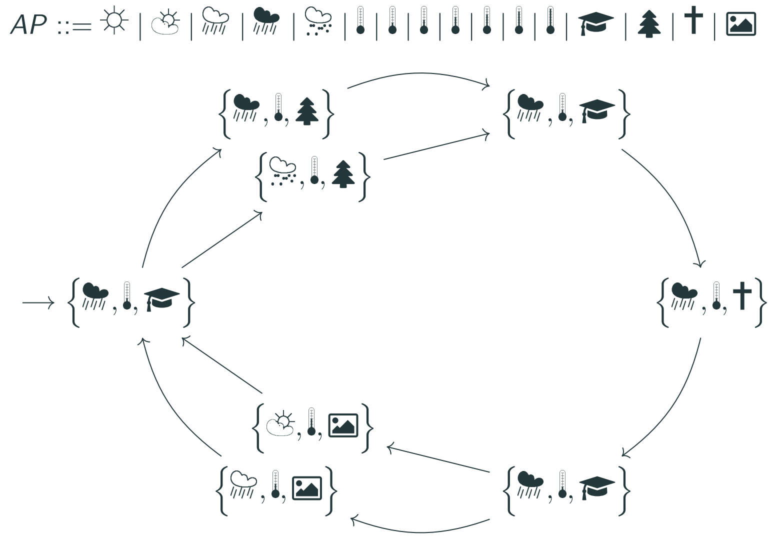
Elements of S are denoted s.

⚠ We interpret not being labelled as a lack of information, not as a negation.

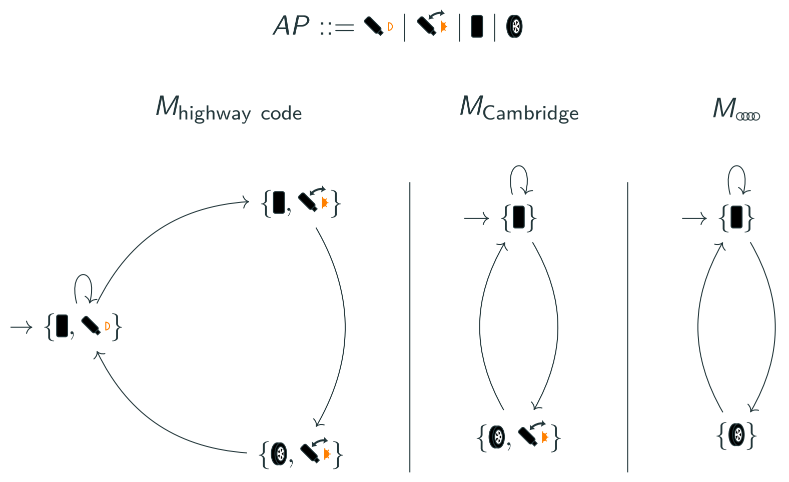
Temporal model of traffic lights



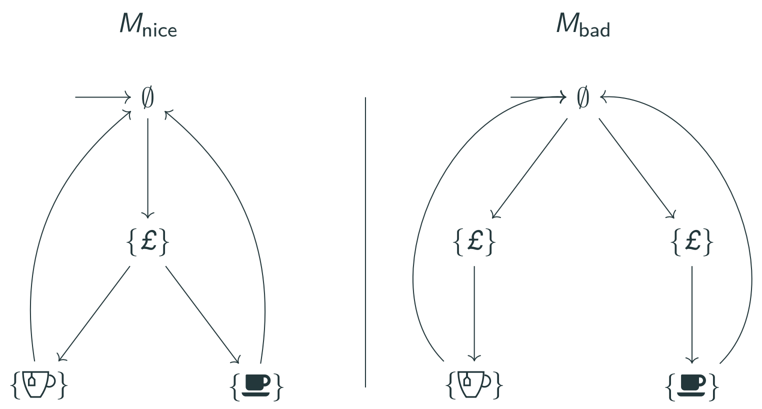
Temporal model of Cambridge weather



Temporal models of indicating



Milner's tea & coffee machines



Corner case: the initial temporal model

$$\mathbb{0} \in \text{TModel}$$

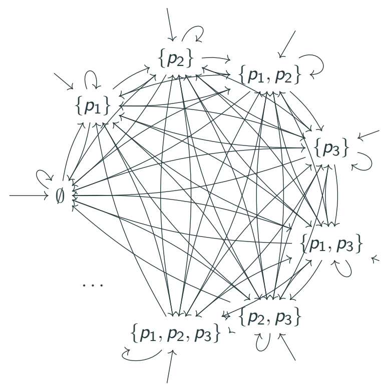
$$\mathbb{0} \overset{\text{def}}{=} \left\langle \begin{array}{l} \mathbb{0}, \\ \emptyset, \\ \emptyset, \\ s \mapsto \emptyset, \\ \ldots \end{array} \right\rangle$$

(it is empty)

Corner case: the terminal temporal model

$$\mathbb{1} \in \text{TModel}$$

$$\mathbb{1} \overset{\text{def}}{=} \left\langle \begin{array}{l} AP \to \mathbb{B}, \\ \{s \mid \top\}, \\ \{s_0, s_1 \mid \top\}, \\ s \mapsto \{p \mid s\ p\}, \\ \ldots \end{array} \right\rangle$$



Temporal model of a terrible punter

A punter with no concern for goat welfare or cabbage welfare:

$$AP = \text{State}$$

$$M = \left\langle \begin{array}{l} \text{State}, \\ \left\{ s \,\middle|\, \forall i.\, s\ i = \cdot\rangle\!\rangle \right\}, \\ \left\{ s, s' \,\middle|\, \left(\begin{array}{l} s\ \text{⛵} = \text{flip}\ (s'\ \text{⛵}) \land \\ (\text{moveone}\ s\ s' \lor \text{movezero}\ s\ s') \end{array} \right) \right\}, \\ s \mapsto \{i \mid s = i\}, \\ \ldots \end{array} \right\rangle$$

Temporal model of a terrible punter

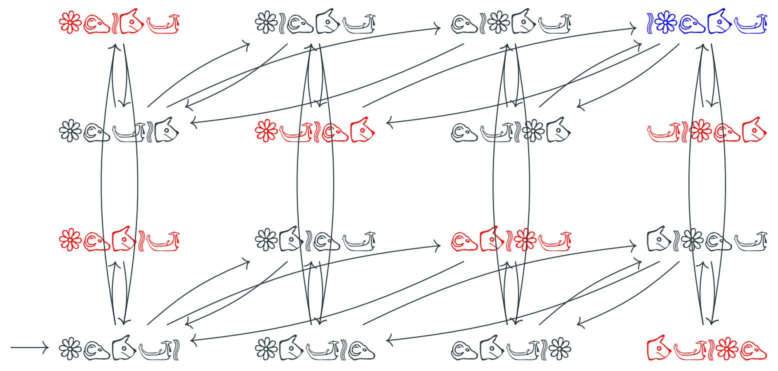
$$\text{flip}\ \cdot\rangle\!\rangle \overset{\text{def}}{=} \rangle\!\rangle\cdot \quad \text{flip}\ \rangle\!\rangle\cdot \overset{\text{def}}{=} \cdot\rangle\!\rangle$$

$$\text{moveone}\ s\ s' \overset{\text{def}}{=} \left(\begin{array}{l} \text{aux}\ s\ s'\ \text{❀}\ \text{🐏}\ \text{🐺} \lor \\ \text{aux}\ s\ s'\ \text{🐏}\ \text{❀}\ \text{🐺} \lor \\ \text{aux}\ s\ s'\ \text{🐺}\ \text{❀}\ \text{🐏} \end{array} \right)$$

$$\text{aux}\ s\ s'\ a\ b\ c \overset{\text{def}}{=} \left(\begin{array}{l} s\ a = s\ \text{⛵} \land s'\ a = s'\ \text{⛵} \land \\ s'\ b = s\ b \land s'\ c = s\ c \end{array} \right)$$

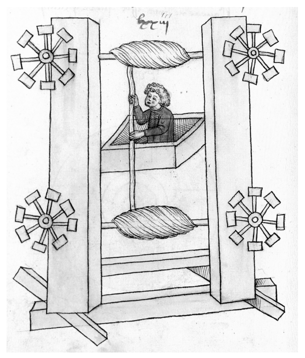
$$\text{movezero}\ s\ s' \overset{\text{def}}{=} s\ \text{❀} = s'\ \text{❀} \land s\ \text{🐏} = s'\ \text{🐏} \land s\ \text{🐺} = s'\ \text{🐺}$$

Temporal model of a terrible punter



Safety: we never go through a red state.
Liveness: we eventually reach the blue state.
Both are pretty clearly false! :-(

Informal temporal model of an elevator



Let us try to describe how an elevator for a building with 3 levels works:

- it starts at the ground floor, with the door closed, and goes back there when it is not called;
- if going through a level where it is called, it stops there and opens its door;
- . . .

Textual descriptions do not scale very well.

Temporal model of an elevator: statics and specification

Direction ::= home | up | down
Level ::= 0 | 1 | 2
Location ::= 0 | 1/2 | 1 | 3/2 | 2
Called $\overset{def}{=}$ Level $\to \mathbb{B}$
DoorStatus ::= open | closed
ElevatorStatus $\overset{def}{=}$ Direction $\times$ Location $\times$ Called $\times$ DoorStatus

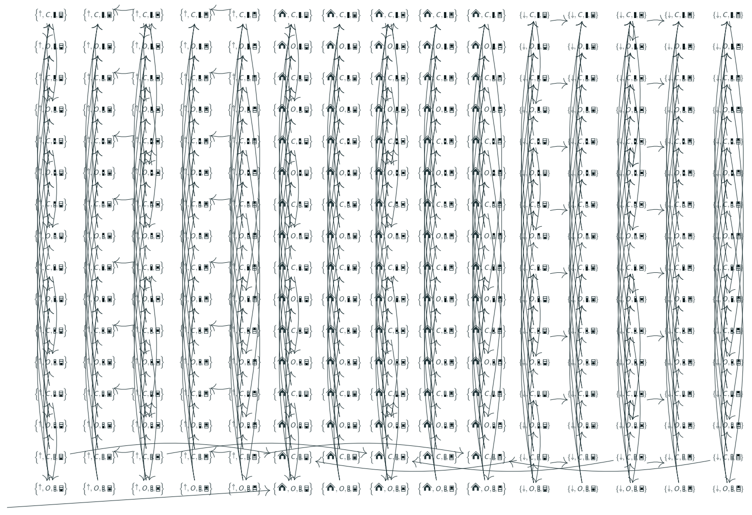
Desired goals:

- the door is not open at half-levels;
- if the elevator is called to a level, then it eventually gets there;
- the elevator does not lock people in;
- the path of the elevator is not entirely idiotic.

Temporal model of an elevator: partial dynamics

$$\begin{tikzcd}
{} \arrow[r] & \text{start} \arrow[d, shift left, "\text{call 0}"] \arrow[rd, "\text{call 1}"] & & \\
& \text{open door} \arrow[u, shift left, "\text{time}"] & \text{going up} \arrow[d, "\text{time}"] \arrow[r, "\text{call 0}"] & \text{reopen door} \arrow[l, bend left, "\text{time}"] \\
& & \text{moving up} \arrow[d, "\text{time}"] & \\
& & \text{arrived} \arrow[d, "\text{time}"] & \\
& & \text{open door} \arrow[d, "\text{time}"] & \\
& & \text{going down} \arrow[d, "\text{time}"] & \\
& & \text{moving down} \arrow[uuuuuu, bend left, "\text{time}"] & \cdots
\end{tikzcd}$$

Temporal model of an elevator: complete (?) dynamics



How to have any confidence that this is correct?   house by Petr Olsšák

Definitions

(Infinite) Paths

$$\mathsf{stream} \in \mathsf{Set} \to \mathsf{Set}$$
$$\pi, \ldots \in \mathsf{stream}\ A \overset{\mathit{def}}{=} \mathbb{N} \to A$$

$$\mathsf{IsPath} \in (M \in \mathsf{TModel}) \to \mathsf{stream}\ M.S \to \mathsf{Prop}$$
$$\mathsf{IsPath}\ M\ \pi \overset{\mathit{def}}{=} \forall n \in \mathbb{N}. (\pi\ n)\ M.T\ (\pi\ (n+1))$$

$$\mathsf{Path} \in \mathsf{TModel} \to \mathsf{Set}$$
$$\mathsf{Path}\ M \overset{\mathit{def}}{=} (\pi \in \mathsf{stream}\ M.S) \times \mathsf{IsPath}\ M\ \pi$$

Reachable states & the tail operation

Because the transition relation is left-total, these infinite paths are “complete”, in the sense that they capture reachability:

$$\mathsf{Reachable} \in (M \in \mathsf{TModel}) \to M.S \to \mathsf{Prop}$$
$$\mathsf{Reachable}\ M\ s \overset{\mathit{def}}{=}\ \exists \pi \in \mathsf{stream}\ M.S, n \in \mathbb{N}.$$
$$\mathsf{IsPath}\ M\ \pi \wedge M.S_0\ (\pi\ 0) \wedge s = \pi\ n$$

$$\mathsf{tailn} \in (A \in \mathsf{Set}) \to \mathbb{N} \to \mathsf{stream}\ A \to \mathsf{stream}\ A$$
$$\mathsf{tailn}\ A\ n\ \pi \overset{\mathit{def}}{=} i \mapsto \pi\ (i+n)$$

Stuttering

A temporal model is **stuttering** when all states loop back to themselves:

$$\begin{aligned} &\text{stuttering} \in \text{TModel} \to \text{Prop} \\ &\text{stuttering}\ M \overset{def}{=} \forall s \in M\mathbf{.}S.\ s\ M\mathbf{.}T\ s \end{aligned}$$

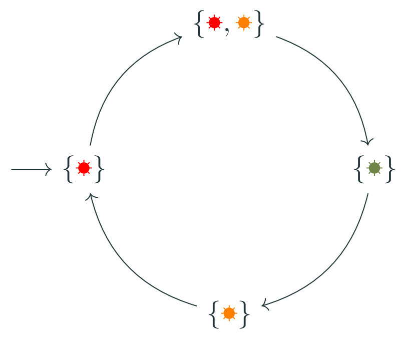
⚠ If the temporal model is not stuttering, then we can count transitions. This is only sound if they exactly match those of the system being analysed.

See “What good is temporal logic” §2.3, by Leslie Lamport
`https://lamport.azurewebsites.net/pubs/what-good.pdf`

More abstract temporal model of traffic lights

Somewhat unusually, we do not interpret a state not being labelled with a given label as that state being labelled with the negation of that label.

AP ::= 🔴 | 🟠 | 🟢



Applications of model checking

Applications of model checking

- Hardware:
  - circuits (with memory) directly translate to temporal models
  - lots of protocols
    - cache protocols
    - bus protocols
    - ...

    their specification involves lots of temporal “liveness” (“eventually something good”) properties
- Software: often not finite a priori, but “proper modelling”, or bounded model-checking
- Security protocols
- Distributed systems
- ...

The common denominator of many of these is the “killer app” of model checking: concurrency.

Examples

In the rest of this lecture, we will sketch how some of these are approached.

The point is not the details of any individual temporal model, but the overall approach.

Temporal model from operational semantics

Temporal model from operational semantics

An initial configuration for a small-step operational semantics naturally leads to a temporal model: take

- configurations as states,
- the initial configuration as the (only) initial state,
- steps as transitions, and
- some interesting properties as atomic propositions, for example

$$
\begin{array}{lcl}
X, Y, Z, \ldots & \in & \mathrm{Var} \\
v & \in & \mathbb{Z} \\
AP & ::= & X \doteq v \mid X \doteq Y \mid X \mathrel{\dot{<}} Y \mid \\
 & & X \mathbin{\dot{+}} Y \mathrel{\dot{<}} Z \mid X \mathbin{\dot{\times}} Y \mathrel{\dot{<}} Z \mid \\
 & & \ldots
\end{array}
$$

Temporal model from operational semantics

For example, for a language with a concurrent composition with interleaving dynamics (as in lecture 6):

$$\begin{tikzcd}
 & & \langle C_1' || C_2 || C_3', s_e \rangle & \\
 & \langle C_1' || C_2 || C_3, s_b \rangle \arrow[ur] \arrow[r] \arrow[dr] & \langle C_1'' || C_2 || C_3, s_f \rangle & \\
 & & \langle C_1' || C_2' || C_3, s_g \rangle & \\
 & & \langle C_1' || C_2' || C_3, s_h \rangle & \\
\langle C_1 || C_2 || C_3, s_a \rangle \arrow[uuur] \arrow[r] \arrow[dddr] & \langle C_1 || C_2' || C_3, s_c \rangle \arrow[ur] \arrow[r] \arrow[dr] & \langle C_1 || C_2'' || C_3, s_i \rangle & \cdots \\
 & & \langle C_1 || C_2' || C_3', s_h \rangle & \\
 & & \langle C_1 || C_2' || C_3', s_k \rangle & \\
 & \langle C_1 || C_2 || C_3', s_d \rangle \arrow[ur] \arrow[r] \arrow[dr] & \langle C_1 || C_2 || C_3'', s_l \rangle & \\
 & & \langle C_1' || C_2 || C_3', s_m \rangle &
\end{tikzcd}$$

Dealing with the size of temporal model from operational semantics

These temporal models are very often infinite or intractably large!

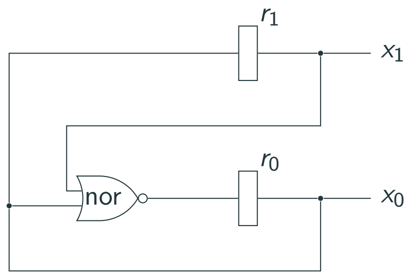
Many approaches:

- bounded model checking (see lecture 9):
  - assume (and possibly check whether) loops execute no more than n times
  - consider executions of length smaller than n
  - ...
- use a model checking DSL to write an idealised version of the program (see lecture 9)
- use abstraction (see lecture 10)

Temporal model from circuits

Example circuit

Synchronous (the clock is left implicit) counter that goes $0, 1, 2, 0, 1, 2, \dots$ (assuming all registers are initially 0):



Registers make the circuit not be a simple function, which motivates using a temporal model.

Example circuit temporal model

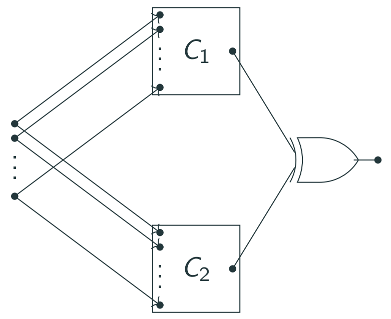
The states of the temporal model are the state of the registers, and the labels are which registers are set to 1:

$$\begin{tikzcd}
{} \arrow[rd] & & \\
& \emptyset \arrow[r] & \{r_0\} \arrow[d] \\
& \{r_0, r_1\} \arrow[r] & \{r_1\} \arrow[lu]
\end{tikzcd}$$

Safety: The state $\{r_0, r_1\}$ should never be reached.
Liveness: all other states should be visited infinitely often.

Difference circuit

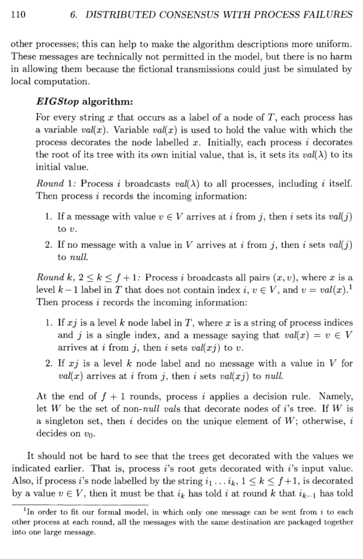
Given two circuits $C_1, C_2 \in \text{SCircuit } i \; 1$, we can define their difference circuit $C_1 \ominus C_2$:



If the answer is always 0, then they are equivalent (see lecture 10). The typical use case is to have a simple, clearly correct C_1, and a complex C_2 to verify.

Temporal models of distributed algorithms

Temporal models of distributed algorithms



other processes; this can help to make the algorithm descriptions more uniform. These messages are technically not permitted in the model, but there is no harm in allowing them because the fictional transmissions could just be simulated by local computation.

***EIGStop* algorithm:**

For every string x that occurs as a label of a node of T, each process has a variable $val(x)$. Variable $val(x)$ is used to hold the value with which the process decorates the node labelled x. Initially, each process i decorates the root of its tree with its own initial value, that is, it sets its $val(\lambda)$ to its initial value.

Round 1*:* Process i broadcasts $val(\lambda)$ to all processes, including i itself. Then process i records the incoming information:

1. If a message with value $v \in V$ arrives at i from j, then i sets its $val(j)$ to v.
2. If no message with a value in V arrives at i from j, then i sets $val(j)$ to *null*.

Round k, $2 \leq k \leq f+1$*:* Process i broadcasts all pairs (x, v), where x is a level $k-1$ label in T that does not contain index i, $v \in V$, and $v = val(x)$.[1] Then process i records the incoming information:

1. If xj is a level k node label in T, where x is a string of process indices and j is a single index, and a message saying that $val(x) = v \in V$ arrives at i from j, then i sets $val(xj)$ to v.
2. If xj is a level k node label and no message with a value in V for $val(x)$ arrives at i from j, then i sets $val(xj)$ to *null*.

At the end of $f+1$ rounds, process i applies a decision rule. Namely, let W be the set of non-*null* *val*s that decorate nodes of i's tree. If W is a singleton set, then i decides on the unique element of W; otherwise, i decides on v_0.

It should not be hard to see that the trees get decorated with the values we indicated earlier. That is, process i's root gets decorated with i's input value. Also, if process i's node labelled by the string $i_1 \ldots i_k$, $1 \leq k \leq f+1$, is decorated by a value $v \in V$, then it must be that i_k has told i at round k that i_{k-1} has told

[1] In order to fit our formal model, in which only one message can be sent from i to each other process at each round, all the messages with the same destination are packaged together into one large message.

Nodes in distributed algorithms are often specified in terms of interacting automata; the temporal model directly results from their interaction.

See IB Concurrent and Distributed Systems

Distributed Algorithms, by Nancy Lynch.

Models of cache algorithms

Models of cache algorithms

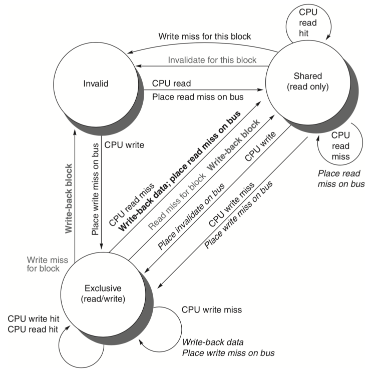


Figure 5.7 Cache coherence state diagram with the state transitions induced by the local processor shown in black and by the bus activities shown in gray. As in Figure 5.6, the activities on a transition are shown in bold.

Cache algorithms are also often specified in terms of interacting automata (they are distributed algorithms too).

See Section 21.5.2.1 German's Protocol in the Handbook of Model Checking.

Computer Architecture: A Quantitative Approach, by Hennessy & Patterson.

Models of security protocols

Models of security protocols

Given a security protocol, define a temporal model where a state contains:

- the state of each agent
- the set of messages sent
- the set of all the messages that can be deduced from the messages sent; this includes taking messages apart, and reassembling them, including via hashing or encrypting using known keys

and where there is a transition from one state to another when

- an agent sends a message
- an adversary sends a deducible message to an agent

See Chapter 22 Model Checking Security Protocols, in the Handbook of Model Checking.

Remark on examples

As illustrated, interesting programs are big, often too big to work on by hand.

This is why we use model checkers, but it also means we cannot easily work with these examples.

Instead, we will mostly look at games and puzzles like the cabbage-goat-wolf puzzle.

Summary

Temporal models make it possible to describe systems that evolve in time.

Temporal models can be extracted directly, for example from circuits, or hand-crafted to capture the relevant parts of an artefact.

In the next lecture, we will see how to use temporal logic(s) to specify the behaviour of temporal models.

Hoare logic and Model checking

Part II: Model checking

Lecture 8: Temporal logic

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2020/21

Recap

In the previous lecture, we saw how temporal models can be used to model various systems.

In this lecture, we will look at how temporal logic can be used to specify the behaviour of temporal models.

Why not use first-order logic?

Why not model time explicitly in first-order logic with equality and $<$, and have variables represent time points?

For example:

$$\forall t_1 .\, p(t_1) \Rightarrow (\exists t_2 .\, t_1 < t_2 \wedge q(t_2))$$

✓ It works.
✓ It has a well-understood theory.
✗ It is very error-prone.
✗ Is is very expensive to check.

Paths and states

Two intuitive things to consider: states, and paths.

- CTL* allows both,
- CTL (computation tree logic) focuses on states,
- LTL (linear temporal logic) focuses on paths.

When using model checking, one generally picks (a language based on) either LTL or CTL.
To describe model checking, CTL* makes things clearer.

We will first focus on the implication-free fragment.

Syntax of the implication-free fragment of CTL*

Given a fixed set of atomic propositions AP,

$\psi, \ldots \in$ StateProp ::=

$\bot$	|	false
$\top$	|	true
$\psi_1 \wedge^{\mathsf{s}} \psi_2$	|	conjunction
$\psi_1 \vee^{\mathsf{s}} \psi_2$	|	disjunction
injp p	|	atomic predicate
A ϕ	|	universal
E ϕ		existential

$\phi, \ldots \in$ PathProp ::=

$\phi_1 \wedge^{\mathsf{p}} \phi_2$	|	conjunction
$\phi_1 \vee^{\mathsf{p}} \phi_2$	|	disjunction
injs ψ	|	state property
X ϕ	|	next
F ϕ	|	future
G ϕ	|	generally
ϕ_1 U ϕ_2		until

We almost always omit injp and injs.

Informal semantics of the implication-free fragment of CTL*

- injp p: the current state satisfies atomic proposition p
- A ϕ: all paths starting from the current state satisfy ϕ
- E ϕ: some path starting from the current state satisfies ϕ
- injs ψ: the first state of the current path satisfies ψ
- G ϕ: every suffix of the current path satisfies ϕ
- F ϕ: some suffix of the current path satisfies ϕ
- X ϕ: the tail of the current path satisfies ϕ
- ϕ_1 U ϕ_2: some suffix of the current path satisfies ϕ_2, and all the suffixes of the current path of which that path is a suffix satisfy ϕ_1

Example propositions in the implication-free fragment of CTL*

- E (F (injs (injp p))): there is a state reachable from the current state that satisfies atomic proposition p
- E (F (injs (injp p $\wedge^{\mathsf{s}}$ injp q))): there is a state reachable from the current state that satisfies both atomic proposition p and atomic proposition q
- (E (F (injs (injp p)))) $\wedge^{\mathsf{s}}$ (E (F (injs (injp q)))): there is a state reachable from the current state that satisfies atomic proposition p, and a reachable state that satisfies proposition q
- E ((F (injs (injp p))) $\wedge^{\mathsf{p}}$ (F (injs (injp q)))): there is a path from the current state, along which there is a state satisfying atomic proposition p, and a state satisfying atomic proposition q
- E (X (injs (injp p))): there is a successor state satisfying atomic proposition p

Example propositions in the implication-free fragment of CTL*

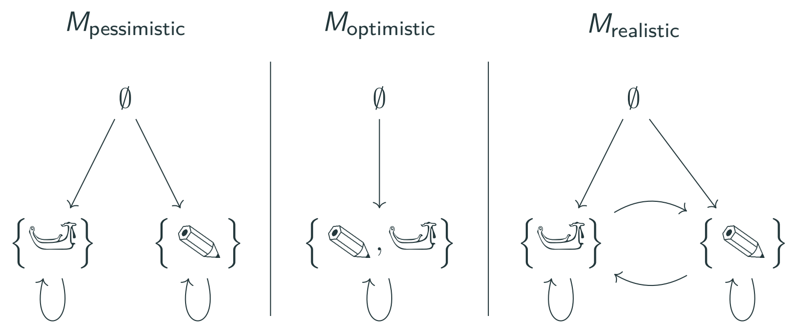
- A (G (injp *p*)): *p* always holds (in any path)
- E (G (injp *p*)): there is one path where *p* always holds
- A (G (A (F (injp idle)))): the tea & coffee machine always goes back to an idle state
- A (F (A (G (injp broken)))): the tea & coffee machine ends up permanently broken

Path conjunction vs. state conjunction

- E (F (injs ((injp *p*) $\wedge^{s}$ (injp *q*)))): there is a state that is reachable from the current state and that satisfies both *p* and *q*
- E ((F (injs (injp *p*))) $\wedge^{p}$ (F (injs (injp *q*)))): there is a state that is reachable from the current state and that satisfies *p*, and a state reachable that is reachable from the current state and that satisfies *q*

Example of path conjunction vs. state conjunction

"At Cambridge, you can row and study"



Semantics of the implication-free fragment of CTL*

We define whether M satisfies ψ,

$$\textcircled{1} \vDash \textcircled{2} \quad \in \mathsf{TModel} \to \mathsf{StateProp} \to \mathsf{Prop}$$
$$M \vDash \psi \overset{def}{=} \forall s \in M.S.\ \ M.S_0\ s \to s \vDash_M \psi$$

using two auxiliary mutually inductive predicates

$$\textcircled{2} \vDash^{\mathsf{s}}_{\textcircled{1}} \textcircled{3} \in (M \in \mathsf{TModel}) \to M.S \to \mathsf{StateProp} \to \mathsf{Prop}$$
$$\textcircled{2} \vDash^{\mathsf{p}}_{\textcircled{1}} \textcircled{3} \in (M \in \mathsf{TModel}) \to \mathsf{stream}\ M.S \to \mathsf{PathProp} \to \mathsf{Prop}$$

We write the arguments that remain constant through recursive calls in this shade of grey blue.

Semantics of the implication-free fragment of CTL*: state properties

$$
\begin{array}{lll}
s \models^{\mathsf{s}}_{M} \top & \overset{def}{=} & \top \\
s \models^{\mathsf{s}}_{M} \bot & \overset{def}{=} & \bot \\
s \models^{\mathsf{s}}_{M} \psi_1 \wedge^{\mathsf{s}} \psi_2 & \overset{def}{=} & (s \models^{\mathsf{s}}_{M} \psi_1) \wedge (s \models^{\mathsf{s}}_{M} \psi_2) \\
s \models^{\mathsf{s}}_{M} \psi_1 \vee^{\mathsf{s}} \psi_2 & \overset{def}{=} & (s \models^{\mathsf{s}}_{M} \psi_1) \vee (s \models^{\mathsf{s}}_{M} \psi_2) \\
s \models^{\mathsf{s}}_{M} \mathsf{injp}\ p & \overset{def}{=} & M.\ell\ s\ p \\
s \models^{\mathsf{s}}_{M} \mathsf{A}\ \phi & \overset{def}{=} & \left(\begin{array}{l} \forall \pi \in \mathsf{stream}\ M.S. \\ \quad \mathsf{IsPath}\ M\ \pi \to \pi\ 0 = s \to \pi \models^{\mathsf{p}}_{M} \phi \end{array}\right) \\
s \models^{\mathsf{s}}_{M} \mathsf{E}\ \phi & \overset{def}{=} & \left(\begin{array}{l} \exists \pi \in \mathsf{stream}\ M.S. \\ \quad \mathsf{IsPath}\ M\ \pi \wedge \pi\ 0 = s \wedge \\ \quad \pi \models^{\mathsf{p}}_{M} \phi \end{array}\right)
\end{array}
$$

Semantics of the implication-free fragment of CTL*: path properties

$$
\begin{array}{lll}
\pi \models^{\mathsf{p}}_{M} \mathsf{injs}\ \psi & \overset{def}{=} & (\pi\ 0) \models^{\mathsf{s}}_{M} \psi \\
\pi \models^{\mathsf{p}}_{M} \phi_1 \wedge^{\mathsf{p}} \phi_2 & \overset{def}{=} & (\pi \models^{\mathsf{p}}_{M} \phi_1) \wedge (\pi \models^{\mathsf{p}}_{M} \phi_2) \\
\pi \models^{\mathsf{p}}_{M} \phi_1 \vee^{\mathsf{p}} \phi_2 & \overset{def}{=} & (\pi \models^{\mathsf{p}}_{M} \phi_1) \vee (\pi \models^{\mathsf{p}}_{M} \phi_2) \\
\pi \models^{\mathsf{p}}_{M} \mathsf{X}\ \phi & \overset{def}{=} & (\mathsf{tailn}\ M.S\ 1\ \pi) \models^{\mathsf{p}}_{M} \phi \\
\pi \models^{\mathsf{p}}_{M} \mathsf{F}\ \phi & \overset{def}{=} & \exists n \in \mathbb{N}.\,(\mathsf{tailn}\ M.S\ n\ \pi) \models^{\mathsf{p}}_{M} \phi \\
\pi \models^{\mathsf{p}}_{M} \mathsf{G}\ \phi & \overset{def}{=} & \forall n \in \mathbb{N}.\,(\mathsf{tailn}\ M.S\ n\ \pi) \models^{\mathsf{p}}_{M} \phi \\
\pi \models^{\mathsf{p}}_{M} \phi_1\ \mathsf{U}\ \phi_2 \overset{def}{=} & &
\end{array}
$$

$$
\exists n \in \mathbb{N}. \left(\begin{array}{l} (\forall k \in \mathbb{N}.\, 0 \le k < n \to (\mathsf{tailn}\ M.S\ k\ \pi) \models^{\mathsf{p}}_{M} \phi_1) \wedge \\ (\mathsf{tailn}\ M.S\ n\ \pi) \models^{\mathsf{p}}_{M} \phi_2 \end{array}\right)
$$

Moving goats

If we extend our atomic propositions to include fine-grained descriptions of the different items, we can write:

$$
\begin{array}{l}
\mathit{Safe} \wedge^{\mathsf{s}} \mathit{Live} \\
\mathit{Safe} \overset{def}{=} \mathsf{A}\ (\mathsf{G}\ \mathit{StateSafe}) \\
\mathit{Live} \overset{def}{=} \mathsf{A}\ (\mathsf{F}\ (\mathit{Done})) \\
\mathit{StateSafe} \overset{def}{=} \text{❀}\text{-}\mathit{Safe} \wedge^{\mathsf{s}} \text{🐏}\text{-}\mathit{Safe} \\
\text{❀}\text{-}\mathit{Safe} \overset{def}{=} \begin{array}{l} (\text{❀🐏🛶≀}) \vee^{\mathsf{s}} (\text{≀❀🐏🛶}) \vee^{\mathsf{s}} \\ (\text{❀≀🐏}) \vee^{\mathsf{s}} (\text{🐏≀❀}) \end{array} \\
\text{🐏}\text{-}\mathit{Safe} \overset{def}{=} \ldots \\
\mathit{Done} \overset{def}{=} \text{≀❀🐏🐺🛶}
\end{array}
$$

We can also express the fact that the puzzle has a solution with

$$
\mathsf{E}\ ((\mathsf{G}\ \mathit{StateSafe}) \wedge^{\mathsf{p}} (\mathsf{F}\ \mathit{Done}))
$$

Informal specification of indicating

Rule 103
Signals warn and inform other road users, including pedestrians ([...]), of your intended actions. You should always

- give clear signals in plenty of time, having checked it is not misleading to signal at that time
- use them to advise other road users before changing course or direction, stopping or moving off
- [...]

Formal specification of indicating

A (G (▮ $\vee$ (▮ U ▶)))

A G ($\mathit{SpecSN} \vee^{p} \mathit{SpecSI} \vee^{p} \mathit{SpecTI}$)
SpecSN = ((▮ $\wedge^{s}$ ▷) $\wedge^{p}$ ((▮ $\wedge^{s}$ ▷) U (▮ $\wedge^{s}$ ▶)))
SpecSI = ((▮ $\wedge^{s}$ ▶) $\wedge^{p}$ ((▮ $\wedge^{s}$ ▶) U (⊛ $\wedge^{s}$ ▶)))
SpecTI = ((⊛ $\wedge^{s}$ ▶) $\wedge^{p}$ ((⊛ $\wedge^{s}$ ▶) U (▮ $\wedge^{s}$ ▷)))

If we want to allow cancelling: make the RHS of until in *SpecSI* have . . . $\vee^{s}$ (▮ $\wedge^{s}$ ▷).
If we want to allow driving straight forever: make the RHS of *SpecSN* have . . . $\vee^{s}$ G (▮ $\wedge^{s}$ ▷); similarly for turning forever.

Implication

Unstable assertions

It may be more natural to use the following:

A (G (⊛ $\rightarrow$ ▶))
A (G ((E (X ⊛)) $\rightarrow$ ▶))

but implication is not stable under abstraction.

We can add implication (and thereby negation) to our temporal logic:

✓ more intuitive,

✗ more brittle: it conflates not being labelled with p with being labelled with $\neg p$, and thus does not respect abstraction.

Syntax of CTL* with implication

StateProp $\in$ Set
$\psi, \ldots \in$ StateProp ::=
$\bot \mid \top \mid \neg^{s}\, \psi \mid \psi_1 \wedge^{s} \psi_2 \mid \psi_1 \vee^{s} \psi_2 \mid \psi_1 \rightarrow^{s} \psi_2 \mid$
injp p | A ϕ | E ϕ

PathProp $\in$ Set
$\phi, \ldots \in$ PathProp ::=
$\neg^{p}\, \phi \mid \phi_1 \wedge^{p} \phi_2 \mid \phi_1 \vee^{p} \phi_2 \mid \phi_1 \rightarrow^{p} \phi_2 \mid$
injs ψ | X ϕ | F ϕ | G ϕ | ϕ_1 U ϕ_2

Splitting

Checking a full CTL* property can be reduced to checking an implication-free CTL* property.

To do so, we need to represent the fact that an atomic proposition can be negated.
We do this by having two versions of each atomic property p:
$\oplus p$ corresponds to p, and $\ominus p$ corresponds to $\neg p$:

$$\begin{aligned}
&\text{Inductive split } (AP \in \text{Set}) \in \text{Set} := \\
&\quad \oplus\text{-} \in AP \to \text{split } AP \\
&\mid \ominus\text{-} \in AP \to \text{split } AP
\end{aligned}$$

Fragments

Fragments

Fragments of CTL*: CTL, LTL, ACTL*, ECTL*, ...

Fragments: CTL

CTL

CTL restricts CTL* so that path quantifiers and temporal operators always come together:

$$
\begin{aligned}
&\psi, \ldots \in \mathsf{StateProp} \in \mathsf{Set} ::= \\
&\quad \bot \mid \top \mid \neg^{\mathsf{s}}\ \psi \mid \\
&\quad \psi_1 \wedge^{\mathsf{s}} \psi_2 \mid \psi_1 \vee^{\mathsf{s}} \psi_2 \mid \psi_1 \rightarrow^{\mathsf{s}} \psi_2 \mid \\
&\quad \mathsf{injp}\ p \mid \mathsf{A}\ \phi \mid \mathsf{E}\ \phi \\
&\phi, \ldots \in \mathsf{PathProp} \in \mathsf{Set} ::= \\
&\quad \mathsf{X}\ (\mathsf{injs}\ \psi) \mid \mathsf{F}\ (\mathsf{injs}\ \psi) \mid \mathsf{G}\ (\mathsf{injs}\ \psi) \mid (\mathsf{injs}\ \psi_1)\ \mathsf{U}\ (\mathsf{injs}\ \psi_2)
\end{aligned}
$$

Limits of CTL

$$
\neg \left(\begin{aligned} &\forall p \in AP.\, \exists \psi^{\mathsf{CTL}} \in \mathsf{StateProp}^{\mathsf{CTL}}. \\ &\quad \forall M \in \mathsf{TModel}.\, (M \vDash \mathsf{F}\ (\mathsf{G}\ p)) \leftrightarrow (M \vDash \psi^{\mathsf{CTL}}) \end{aligned} \right)
$$

Fragments: LTL

LTL

LTL restricts CTL* so that properties are only (universally quantified) path properties:

$$
\begin{aligned}
&\psi, \ldots \in \mathsf{StateProp} ::= \mathsf{A}\ \phi \\
&\phi, \ldots \in \mathsf{PathProp} ::= \\
&\quad \neg^{\mathsf{p}}\ \phi \mid \phi_1 \wedge^{\mathsf{p}} \phi_2 \mid \phi_1 \vee^{\mathsf{p}} \phi_2 \mid \phi_1 \rightarrow^{\mathsf{p}} \phi_2 \mid \\
&\quad \mathsf{injs\text{-}injp}^{\mathsf{WI}}\ p \mid \mathsf{X}\ \phi \mid \mathsf{F}\ \phi \mid \mathsf{G}\ \phi \mid \phi_1\ \mathsf{U}\ \phi_2
\end{aligned}
$$

The leading A is kept implicit.

Limits of LTL

LTL cannot express things like "whenever p holds, it is possible to reach a state where q holds":

$$\neg\left(\begin{array}{l}\forall AP \in \mathsf{Set}.\, \forall p, q \in AP.\, \exists \psi^{\text{LTL}} \in \mathsf{StateProp}^{\text{LTL}}\ AP. \\ \quad \forall M \in \mathsf{TModel}\ AP. \\ \qquad (M \vDash \mathsf{A}\ (\mathsf{G}\ (p \rightarrow \mathsf{E}\ (\mathsf{F}\ q)))) \leftrightarrow (M \vDash \psi^{\text{LTL}})\end{array}\right)$$

CTL vs. LTL: Milner's tea & coffee machines

These can be used to tell the difference between CTL (can distinguish) and LTL (cannot distinguish), because their difference is about their branching structure

Limits of CTL*

CTL* cannot express things like "p holds at even steps".

CTL* also has lots of moving parts. The linear μ-calculus (itself a special case of the modal μ-calculus) is more expressive than CTL*, and has far fewer moving parts, but is quite fiddly.

ACTL* and ECTL*

The universal fragment of CTL*, ACTL*, where all E are under odd numbers of negations, and all A are under even numbers of negations, is well-behaved with respect to abstraction (see lecture 11).

ACTL* contains LTL, and ACTL, the intersection of ACTL* and CTL.
ACTL* is dual (for the negation operation) to the existential fragment of CTL*, ECTL*.

Quantifiers

Unlike in Hoare logic, there are no quantifiers, as they would make it difficult to mechanically check properties.

To make up for this, we can use property schemas with big operators or bounded quantifiers, and indexed atomic propositions, which stand for the expanded property.

For example $\bigwedge_{i=0}^{n} p_i$, for $n = 3$, is expanded to $p_1 \wedge p_2 \wedge p_3$.

So is $\bigwedge_{i \in S} p_i$, for $S = \{1, 2, 3\}$.

This is is not as general as quantifiers, as the value of n or S has to be known. Because this is done as a preprocessing phase, and does not change the language of properties.

Summary

Temporal logics can be used to specify temporal models.

In the next lecture, we will look at how model checking is used in practice.

Hoare logic and Model checking

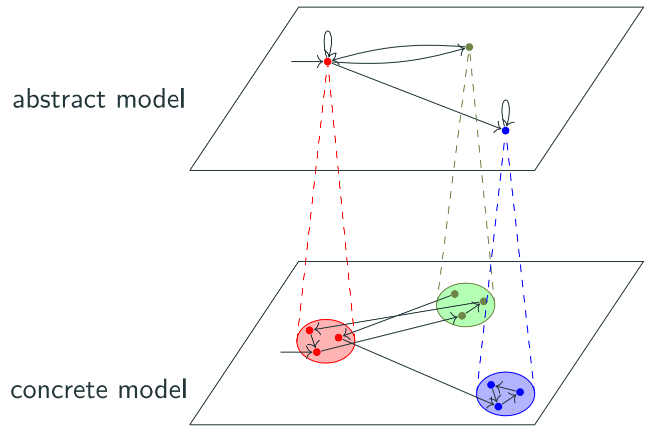
Part II: Model checking

Lecture 10: Relating temporal models

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2020/21

Relating temporal models



Relating temporal models

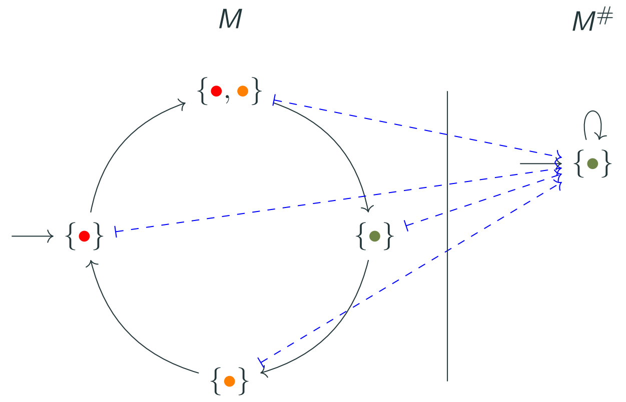
The premise of model checking is that checking the model translates to confidence in the modelled artefact.

If we can express the artefact as a temporal model too, and if the abstract model can **simulate** the concrete model, then we can check some classes of properties on the abstract model and know that they hold of the concrete model.

However, discarding the unimportant aspects the the artefact is also a crucial aspect of modelling.

Abstraction of traffic lights by some Cambridge taxi drivers

$AP ::=$ ● | ● | ●



...still, another crucial aspect of modelling is to not discard the crucial aspects of the artefact.

Temporal model simulation 1/2

R is a **temporal model simulation** of M by M':

$$① \preccurlyeq^{③} ② \in \ (M \in \mathrm{TModel}) \to (M' \in \mathrm{TModel}) \to (M.S \to M'.S \to \mathrm{Prop}) \to \mathrm{Prop}$$

$$M \preccurlyeq^{R} M' \overset{def}{=}$$

(1) R is consistent with labels:

$$\left(\begin{array}{l} \forall s \in M.S, s' \in M'.S. \\ \quad s \ R \ s' \to \forall p \in AP.\, M'.\ell \ s' \ p \to M.\ell \ s \ p \end{array} \right) \wedge$$

(2) R relates initial states of M to initial states in M':

$$(\forall s \in M.S.\, M.S_0 \ s \to \exists s' \in M'.S.\, M'.S_0 \ s' \wedge s \ R \ s') \wedge$$

(continued on the next slide)

Temporal model simulation 2/2

(3) any step in M can be matched by a step in M' from any R-related start state to some R-related end state:

$$\left(\begin{array}{l} \forall s_0, s_1 \in M.S, s'_0 \in M'.S. \\ \quad s_0 \ M.T \ s_1 \wedge s_0 \ R \ s'_0 \to \\ \quad\quad \exists s'_1 \in M'.S. \\ \quad\quad\quad s'_0 \ M'.T \ s'_1 \wedge s_1 \ R \ s'_1 \end{array} \right)$$

$$\begin{tikzcd} s_0 \arrow[r, dashed, "R"] \arrow[d, "M.T"'] & s'_0 & & & s_0 \arrow[r, dashed, "R"] \arrow[d, "M.T"'] & s'_0 \arrow[d, "M'.T"] \\ s_1 & & \to & \exists s'_1. & s_1 \arrow[r, dashed, "R"] & s'_1 \end{tikzcd}$$

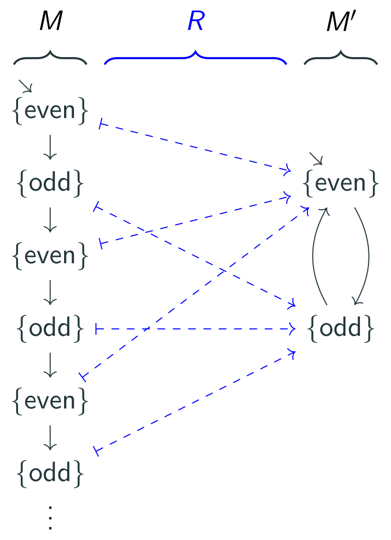
Examples of simulations

The identity relation is a simulation:

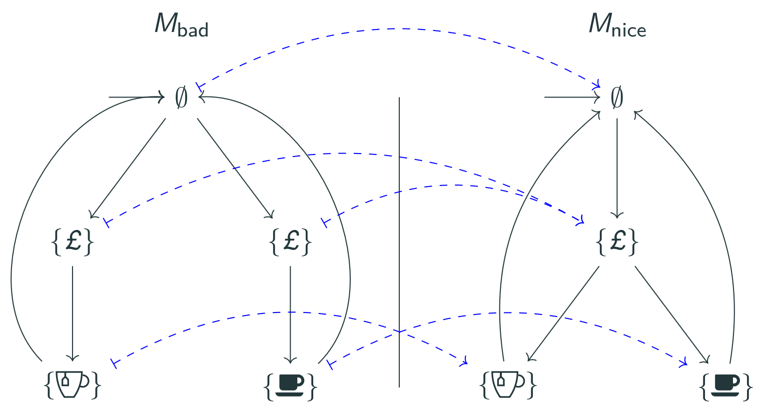
$$\begin{aligned}&\forall M \in \mathsf{TModel}.\\ &\quad \text{let } R = (s \mapsto s) \text{ in}\\ &\quad M \preccurlyeq^{R} M\end{aligned}$$

The terrible punter (lecture 1) can simulate the good punter (lecture 3) by, when it has a choice of things, doing a good thing.

Examples of simulations



Milner's tea & coffee machines



Temporal model simulation

Often, the details of the simulation are not so important, what matters is the existence of a simulation:

$$\begin{aligned}&① \preccurlyeq ② \in \mathsf{TModel} \to \mathsf{TModel} \to \mathsf{Prop}\\ &(M \preccurlyeq M') \overset{\mathit{def}}{=} \exists R.\, M \preccurlyeq^{R} M'\end{aligned}$$

It means that M' is "more abstract" than M: it may have more behaviour, making it less precise, but that allows it to have possibly fewer states and transitions.

Simulation preserves ACTL*

The universal, implication-free fragment of CTL*, ACTL*$^{\mathsf{IF}}$, is compatible with the simulation preorder:

$$\begin{aligned}&\forall M \in \mathsf{TModel}, M' \in \mathsf{TModel}, \psi \in \mathsf{StateProp}^{\mathsf{ACTL^*IF}}.\\&\quad (M \preccurlyeq M' \land \mathsf{us}\ \psi \land M' \vDash \psi) \to M \vDash \psi\end{aligned}$$

(where us ψ means ψ is a universal state property)

It suffices to show the property holds of the more abstract model to know it holds of the more concrete model.

This property can seem strange, because F ϕ has an existential feel to it. In fact, it is very fragile, and really depends on left-totality!

Temporal model bisimulation

R is a **temporal model bisimulation** of M by M':

$$\begin{aligned}① \approx^{③} ② \in\ &(M \in \mathsf{TModel}) \to (M' \in \mathsf{TModel}) \to\\&(M.S \to M'.S \to \mathsf{Prop}) \to \mathsf{Prop}\\M \approx^R M' \overset{def}{=}\ & M \preccurlyeq^R M' \land M' \preccurlyeq^R M\end{aligned}$$

As for simulations, the details of the bisimulation are not so important, often what matters is the existence of a bisimulation:

$$\begin{aligned}&① \approx ② \in \mathsf{TModel} \to \mathsf{TModel} \to \mathsf{Prop}\\&(M \approx M') \overset{def}{=} \exists R.\ M \approx^R M'\end{aligned}$$

Bisimulation preserves CTL*

All of CTL* is compatible with bisimulation equivalence:

$$\begin{aligned}&\forall M \in \mathsf{TModel}, M' \in \mathsf{TModel}, \psi \in \mathsf{StateProp}^{\mathsf{WI}}.\\&\quad M \approx M' \to (M \vDash \psi \leftrightarrow M' \vDash \psi)\end{aligned}$$

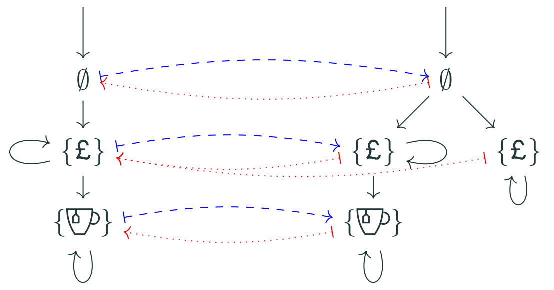
Bisimulation and simulations

Bisimulation implies simulations in both directions

$$M \approx M' \to (M \preccurlyeq M' \land M' \preccurlyeq M)$$

⚠ but in general not the other way around!

For example, on a variation of the tea & coffee machines example:



Revisiting stuttering

What if we want to abstract multiple steps of the concrete model with one step of the abstract model?

$\rightsquigarrow$ We can change our notion of path to allow staying any finite number of times in any state (in addition to allowing forever on states with self-loops).

We can then adapt most of the notions we have seen so far. However, in this setting, we do not want to use the X temporal operator.

This is the approach taken by TLA+.

Summary

We saw how abstraction can be used to relate temporal models in a way that makes checking some classes of properties sound.

...but remember an important part of modelling is judicious under-approximation! $\rightsquigarrow$ domain knowledge is crucial.

In the next lecture, we will look at how to implement model checking.

Hoare logic and Model checking

Part II: Model checking

Lecture 11: Implementing model checking

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2020/21

Definite temporal models

For the model checker to be effective, the input temporal model needs to be effective.

A **definite temporal model**:

$$\begin{aligned}
&\mathrm{DTModel} \in \mathrm{Set}\\
&DM, \ldots \in \mathrm{DTModel} \overset{def}{=}\\
&\quad (S \in \mathrm{Set}) \times\\
&\qquad (F \in \mathrm{Fintype}\ S) \times\\
&\qquad\quad (S_0 \in S \to \mathbb{B}) \times\\
&\qquad\qquad (① \ T \ ② \in S \to S \to \mathbb{B}) \times\\
&\qquad\qquad\quad (\ell \in S \to AP \to \mathbb{B}) \times\\
&\qquad\qquad\qquad (\forall s \in S.\, \exists s' \in S.\, s\ T\ s' = \top_{\mathbb{B}})
\end{aligned}$$

Specifying a CTL model checker

We will see how to implement the world's worst CTL model checker:

$$\mathsf{mc} \in (AP \in \mathsf{Set}) \to \mathsf{DTModel}\ AP \to \mathsf{StateProp}^{\text{CTL}}\ AP \to \mathbb{B}$$

which has the following specification:

$$\begin{aligned}&\forall AP \in \mathsf{Set}, DM \in \mathsf{DTModel}\ AP, \psi^{\text{CTL}} \in \mathsf{StateProp}^{\text{CTL}}\ AP.\\ &\quad \mathsf{reflect}\ (\mathsf{mc}\ AP\ DM\ \psi^{\text{CTL}})\ \ (DM \vDash^{\text{WI}}_{AP} \psi^{\text{CTL}})\end{aligned}$$

where satisfaction in a definite model is as expected.

Defining a CTL model checker

To check whether the model satisfies a property, it suffices to check whether all the initial states satisfy the property, which we check using an auxiliary function *mca* that checks whether a state satisfies a given property.

$$\begin{aligned}&\mathsf{mc}\ AP\ DM\ \psi^{\text{CTL}} \overset{\mathit{def}}{=}\\ &\quad \mathsf{forall\text{-}fin}\ DM.S\ (s \mapsto DM.S_0\ s \to_{\mathbb{B}} \mathsf{mca}\ DM\ \psi^{\text{CTL}}\ s)\end{aligned}$$

$$\begin{aligned}\mathsf{mca} \in\ & (AP \in \mathsf{Set}) \to (DM \in \mathsf{DTModel}\ AP) \to\\ &\quad \mathsf{StateProp}^{\text{CTL}}\ AP \to (DM.S \to \mathbb{B})\end{aligned}$$

This *mca* function works by recursion on the proposition, calling itself on the sub-propositions.

CTL model checker: propositional fragment

$$\begin{aligned}
\mathsf{mca}\ AP\ DM\ p &\overset{\mathit{def}}{=} s \mapsto DM.\ell\ s\ p\\
\mathsf{mca}\ AP\ DM\ (\hat{\neg}\phi^{\text{CTL}}) &\overset{\mathit{def}}{=} \mathsf{let}\ V = \mathsf{mca}\ AP\ DM\ \phi^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ s \mapsto \neg_{\mathbb{B}}(V\ s)\\
\mathsf{mca}\ AP\ DM\ (\phi_1^{\text{CTL}} \mathbin{\hat{\wedge}} \phi_2^{\text{CTL}}) &\overset{\mathit{def}}{=} \mathsf{let}\ V_1 = \mathsf{mca}\ AP\ DM\ \phi_1^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ \mathsf{let}\ V_2 = \mathsf{mca}\ AP\ DM\ \phi_2^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ s \mapsto V_1\ s \wedge_{\mathbb{B}} V_2\ s\\
\mathsf{mca}\ AP\ DM\ (\phi_1^{\text{CTL}} \mathbin{\hat{\vee}} \phi_2^{\text{CTL}}) &\overset{\mathit{def}}{=} \mathsf{let}\ V_1 = \mathsf{mca}\ AP\ DM\ \phi_1^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ \mathsf{let}\ V_2 = \mathsf{mca}\ AP\ DM\ \phi_2^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ s \mapsto V_1\ s \vee_{\mathbb{B}} V_2\ s\\
\mathsf{mca}\ AP\ DM\ (\phi_1^{\text{CTL}} \mathbin{\hat{\to}} \phi_2^{\text{CTL}}) &\overset{\mathit{def}}{=} \mathsf{let}\ V_1 = \mathsf{mca}\ AP\ DM\ \phi_1^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ \mathsf{let}\ V_2 = \mathsf{mca}\ AP\ DM\ \phi_2^{\text{CTL}}\ \mathsf{in}\\
&\quad\ \ s \mapsto V_1\ s \to_{\mathbb{B}} V_2\ s
\end{aligned}$$

CTL model checker: next

If we know in which states ϕ^{CTL} holds, then we know in which states $\mathsf{X}\ \phi^{\text{CTL}}$ holds: their predecessors:

$$\begin{aligned}
&\mathsf{mca}\ AP\ DM\ (\mathsf{A}\ \mathsf{X}\ \phi^{\text{CTL}}) \overset{\mathit{def}}{=}\\
&\quad \mathsf{let}\ V = \mathsf{mca}\ AP\ DM\ \phi^{\text{CTL}}\ \mathsf{in}\\
&\quad s \mapsto \mathsf{forall\text{-}fin}\ DM.S\ (s' \mapsto (s\ DM.T\ s' \to_{\mathbb{B}} V\ s'))\\
&\mathsf{mca}\ AP\ M\ (\mathsf{E}\ \mathsf{X}\ \phi^{\text{CTL}}) \overset{\mathit{def}}{=}\\
&\quad \mathsf{let}\ V = \mathsf{mca}\ AP\ DM\ \phi^{\text{CTL}}\ \mathsf{in}\\
&\quad s \mapsto \mathsf{exists\text{-}fin}\ DM.S\ (s' \mapsto s(DM.T\ s' \wedge_{\mathbb{B}} V\ s'))
\end{aligned}$$

CTL model checker: small paths 1/2

The remaining temporal operators talk about infinite paths.
But it is sufficient to consider paths smaller than the diameter of the model[1]:

$$\begin{aligned}&\mathsf{IsSmallPathFrom} \in (AP \in \mathsf{Set}) \to (DM \in \mathsf{DTModel}\ AP) \to DM.S \to\\ &\qquad \mathsf{list}\ DM.S \to \mathsf{Prop}\\ &\mathsf{IsSmallPathFrom}\ AP\ DM\ s\ \Pi \overset{def}{=}\\ &\quad (\mathsf{length}\ \Pi \leq \mathsf{size}\ DM.F) \wedge (\mathsf{nth}\ \Pi\ 0 = \mathsf{some}\ s) \wedge\\ &\quad (\mathsf{nth}\ \Pi\ (\mathsf{length}\ \Pi - 1) = \mathsf{some}\ s') \wedge (s'\ DM.T\ s) \wedge\\ &\quad \left(\forall n \in \mathbb{N}, s', s''. \begin{pmatrix} \mathsf{nth}\ \Pi\ n = \mathsf{some}\ s' \wedge \\ \mathsf{nth}\ \Pi\ (n+1) = \mathsf{some}\ s'' \end{pmatrix} \to s'\ DM.T\ s'' = \top_{\mathbb{B}}\right)\end{aligned}$$

[1]reminiscent of the pumping lemma for automata.

CTL model checker: small paths 2/2

And we can obtain all these paths:

$$\begin{aligned}&\mathsf{small\text{-}paths\text{-}from} \in (AP \in \mathsf{Set}) \to (DM \in \mathsf{DTModel}\ AP) \to\\ &\qquad (s \in DM.S) \to\\ &\qquad\quad \mathsf{Fintype}\ (\mathsf{SmallPathFrom}\ AP\ DM\ s)\\ &\mathsf{small\text{-}paths\text{-}from} \overset{def}{=} \ldots\end{aligned}$$

CTL model checker: generally

For the ‘generally’ temporal operator, we need to look at lasso-shaped paths that are made up of a loop and a (possibly empty) path that leads to that loop, and check that all the states of this lasso satisfy the sub-property:

```
mca $AP$ $DM$ (A G $\phi^{\text{CTL}}$) $\overset{def}{=}$
  let $V$ = mca $AP$ $DM$ $\phi^{\text{CTL}}$ in
  $s \mapsto$ forall-fin
        (small-paths-from $AP$ $DM$ $s$)
        ($\Pi \mapsto$ forall-list $\Pi$ ($s' \mapsto V\ s'$))
mca $AP$ $DM$ (E G $\phi^{\text{CTL}}$) $\overset{def}{=}$
  let $V$ = mca $AP$ $DM$ $\phi^{\text{CTL}}$ in
  $s \mapsto$ exists-fin
        (small-paths-from $AP$ $DM$ $s$)
        ($\Pi \mapsto$ forall-list $\Pi$ ($s' \mapsto V\ s'$))
```

CTL model checker: future

$$\mathsf{mca}\ AP\ DM\ (\mathsf{A\ F}\ \phi^{\text{CTL}}) \overset{def}{=} \ldots$$

$$\mathsf{mca}\ AP\ DM\ (\mathsf{E\ F}\ \phi^{\text{CTL}}) \overset{def}{=} \ldots$$

Left as an exercise.

CTL model checker: until

```
mca $AP$ $DM$ (A ($\phi_1^{\text{CTL}}$ U $\phi_2^{\text{CTL}}$)) $\overset{def}{=}$
  let $V_1$ = mca $AP$ $DM$ $\phi_1^{\text{CTL}}$ in
  let $V_2$ = mca $AP$ $DM$ $\phi_2^{\text{CTL}}$ in
  $s \mapsto$ (forall-fin (small-paths-from $AP$ $DM$ $s$)
      ($\Pi \mapsto$
          (existi $\Pi$
              ($j\ s'' \mapsto$
                  ((forall $\Pi$ ($i\ s' \mapsto j <_{\mathbb{B}} i \to_{\mathbb{B}} V_1\ s'$))
                   $\wedge_{\mathbb{B}}$ $V_2\ s''$)))))
```

mca AP DM (E (ϕ_1^{CTL} U ϕ_2^{CTL})) $\overset{def}{=} \ldots$

Left as an exercise.

Actually implementing model checking

This is not very efficient!

In practice,

- the Vs are memoised;
- "symbolic model checking" uses binary decision diagrams (IB Logic and proof) to represent sets of states, and performs operations on sets-as-BDDs, instead of explicitly manipulating the sets;
- the states can be computed lazily;
- "partial order reduction" tries to not enumerate redundant interleavings;
- …
- 40+ years of tricks!

Counterexamples

Generating counterexamples

Adapted from "Tree-Like Counterexamples in Model Checking".

If the specification is not satisfied, and is in ACTL, then we can do better than just say "no": we can produce a counterexample.

The idea is that $M \nvDash_{AP} \psi^{\text{ACTL}}$ is equivalent to $M \vDash_{AP} \neg\psi^{\text{ACTL}}$, which is itself equivalent to nf-model $M \vDash_{AP}$ nf-neg$^{\text{s}}$ AP ψ^{ACTL}, where the latter formula is (the embedding of a proposition) in ECTL: it suffices to find a witness of that ECTL proposition.

Shape of ECTL witnesses

The shape of an ECTL witness:

$W, \ldots \in$ data Witness $(AP \in \mathsf{Set})\ (M \in \mathsf{TModel}\ AP) \in \mathsf{Set} :=$
  wap $\in M.S \to$ Witness $AP\ M$
| wand $\in$ Witness $AP\ M \to$ Witness $AP\ M \to$ Witness $AP\ M$
| winjl $\in$ Witness $AP\ M \to$ Witness $AP\ M$
| winjr $\in$ Witness $AP\ M \to$ Witness $AP\ M$
| wX $\in M.S \to M.S \to$ Witness $AP\ M \to$ Witness $AP\ M$
| wF $\in$ list $M.S \to$ Witness $AP\ M \to$ Witness $AP\ M$
| wG $\in$ list $(M.S \times$ Witness $AP\ M) \to$ Witness $AP\ M$
| wU $\in$ list $(M.S \times$ Witness $AP\ M) \to M.S \to$ Witness $AP\ M \to$
      Witness $AP\ M$

Being an ECTL witness: atomic propositions

$\_ \vDash_{\_} \_$ wit-by $\_$: $(AP \in \mathsf{Set}) \to (M \in \mathsf{TModel}\ AP) \to M.S \to$
$(\psi \in \mathsf{StateProp}^{\mathsf{CTL}}\ AP) \to$ Witness $AP\ M\ s \to$
Prop

There are (on purpose) no cases for A ....

A witness for an atomic proposition is just that the atomic proposition holds immediately:

$s \vDash_{AP,M} p$ wit-by $W \overset{def}{=} M.\ell\ s\ p \wedge W =$ wap $AP\ M\ s$

Being an ECTL witness: next

A witness for next is a transition from the current state, and a witness that the sub-property holds from the end state:

$s \vDash_{AP,M}$ E X ψ wit-by $W \overset{def}{=}$

$$\exists s' \in M.S, W' \in \text{Witness } AP\ M. \begin{pmatrix} s\ M.T\ s' \wedge \\ s' \vDash_{AP,M} \psi \text{ wit-by } W' \wedge \\ W = \text{wX } AP\ M\ s\ s'\ W' \end{pmatrix}$$

Being an ECTL witness: future

A witness for the 'future' temporal operator is a path that leads to a state for which we have a witness that it satisfies the sub-property:

$s \vDash_{AP,M}$ E F ψ wit-by $W \overset{def}{=}$

$$\exists s' \in M.S, \Pi \in \text{list } M.S, W' \in \text{Witness } AP\ M. \begin{pmatrix} \text{IsSmallPathFrom } AP\ M\ s\ \Pi \wedge \\ \text{last } \Pi = \text{some } s' \wedge \\ s' \vDash_{AP,M} \psi \text{ wit-by } W' \wedge \\ W = \text{wF } AP\ M\ s\ \Pi\ W' \end{pmatrix}$$

Being an ECTL witness: generally

A witness for the ‘generally’ temporal operator is a lasso, for all the states of which we have a witness that they satisfy the sub-property:

$$
\begin{array}{l}
s \vDash_{AP,M} \mathsf{E}\ \mathsf{G}\ \psi \text{ wit-by } W \overset{def}{=} \\
\quad \text{let } T = (M.S \times \text{Witness } AP\ M) \text{ in} \\
\quad \exists X \in \text{list } T. \\
\qquad \left(\begin{array}{l}
\text{IsSmallPathFrom } AP\ M\ s\ X \wedge \\
(\exists i. (\text{last } T\ X)\ M.T\ (\text{nth } T\ X\ i)) \wedge \\
\left(\begin{array}{l}
\forall i \in \mathbb{N}, s' \in M.S, W' \in \text{Witness } AP\ M\ s'. \\
\quad \left(\begin{array}{l}
\text{nth } T\ X\ i = \text{some } \langle s', W' \rangle \rightarrow \\
\quad s' \vDash_{AP,M} \psi \text{ wit-by } W'
\end{array} \right)
\end{array} \right) \wedge \\
W = \text{wG } AP\ M\ X)
\end{array} \right)
\end{array}
$$

Being an ECTL witness: until

$$
\begin{array}{l}
s \vDash_{AP,M} \mathsf{E}\ \psi_1\ \mathsf{U}\ \psi_2 \text{ wit-by } W \overset{def}{=} \\
\quad \text{let } T = (M.S \times \text{Witness } AP\ M) \text{ in} \\
\quad \exists X \in \text{list } T, s' \in M.S, W' \in \text{Witness } AP\ M. \\
\qquad \left(\begin{array}{l}
\text{IsSmallPathFrom } AP\ M\ s\ (X +\!\!+ [\langle s', W' \rangle]) \wedge \\
\left(\begin{array}{l}
\forall i \in \mathbb{N}, s'' \in M.S, W'' \in \text{Witness } AP\ M\ s'. \\
\quad \left(\begin{array}{l}
\text{nth } T\ X\ i = \text{some } \langle s'', W''' \rangle \rightarrow \\
\quad s'' \vDash_{AP,M} \psi_1 \text{ wit-by } W''
\end{array} \right)
\end{array} \right) \wedge \\
(s' \vDash_{AP,M} \psi_2 \text{ wit-by } W') \wedge \\
W = \text{wU } AP\ M\ X\ s'\ W')
\end{array} \right)
\end{array}
$$

Being an ECTL witness: conjunction

$$
\begin{array}{l}
s \vDash_{AP,M} \psi_1 \hat{\wedge} \psi_2 \text{ wit-by } W \overset{def}{=} \\
\quad \exists W_1 \in \text{Witness } AP\ M, W_2 \in \text{Witness } AP\ M. \\
\quad \left(\begin{array}{l}
s \vDash_{AP,M} \psi_1 \text{ wit-by } W_1 \wedge s \vDash_{AP,M} \psi_2 \text{ wit-by } W_2 \wedge \\
W = \text{wand } AP\ M\ W_1\ W_2
\end{array} \right)
\end{array}
$$

Being an ECTL witness: disjunction

$$
\begin{array}{l}
s \vDash_{AP,M} \psi_1 \hat{\vee} \psi_2 \text{ wit-by } W \overset{def}{=} \\
\quad \left(\begin{array}{l}
\exists W_1 \in \text{Witness } AP\ M. \\
\quad \left(\begin{array}{l}
s \vDash_{AP,M} \psi_1 \text{ wit-by } W_1 \wedge \\
W = \text{winjl } AP\ M\ W_1
\end{array} \right)
\end{array} \right) \vee \\
\quad \left(\begin{array}{l}
\exists W_2 \in \text{Witness } AP\ M. \\
\quad \left(\begin{array}{l}
s \vDash_{AP,M} \psi_2 \text{ wit-by } W_2 \wedge \\
W = \text{winjr } AP\ M\ W_2
\end{array} \right)
\end{array} \right)
\end{array}
$$

Satisfiability and existence of witnesses

The requirement for a DTModel is just a brutal way to require M to be finite (otherwise, the witness could be infinite, and we would need a coinductive definition of a witness)

$$\begin{array}{l}
\forall AP \in \text{Set}, M \in \text{TModel } AP, DM \in \text{DTModel } AP,\\
\quad s \in M.S, \psi \in \text{StateProp}^{\text{CTL}}\ AP.\\
\qquad \text{es } \psi \to \text{reflect-model } AP\ M\ DM \to\\
\qquad\quad \left(\begin{array}{l} (s \vDash^{\text{WI-S}}_{AP,M} \psi) \leftrightarrow \\ \left(\begin{array}{l} \exists W \in \text{Witness (split } AP\text{) (nf-model } AP\ M\text{)}.\\ \quad s \vDash_{(\text{split } AP),(\text{nf-model } AP\ M)} (\text{nf}^{\text{s}}\ AP\ \psi) \text{ wit-by } W \end{array} \right) \end{array} \right)
\end{array}$$

Now, if we have $M \nvDash_{AP} \psi^{\text{ACTL}}$, there exists a corresponding W — and we can effectively find it by tweaking our model checking algorithm above (details elided).

Witnesses beyond ECTL

Can we have witnesses for more than just ECTL?

Yes, for example, one of the nice things about LTL is that counterexamples are just paths.

But if we look at fragments of CTL* that are to expressive, then these witnesses are often difficult to understand and use.

Instead, focus has been mostly on making better counterexamples for common subsets of ECTL.

Model checking LTL and CTL*

Requires a bit of machinery to check whether a state is visited infinitely often: Büchi automata.

We will not consider this further.

Summary

We saw a model checking algorithm for CTL, and sketched how it could be modified to generate counterexamples for ACTL formulas.

CEGAR
not examinable

CEGAR

Assume that we have a way to automatically generate abstract models. Then we can take the following approach: recursively:
pick an abstraction of the model
check the property in the abstract model
if it is true, happy
if it is false, is it a genuine counterexample?
try it on the base model: if it works, we have found a genuine counterexample
if it does not work, build an abstraction.

Model checking hybrid systems

Modelling physical systems is often best done with continuous variables. Is it possible apply model checking to these?

Yes! It has been done for example for ACAS X, the Next-Generation Airborne Collision Avoidance System
`https://doi.org/10.1007/s10009-016-0434-1`

Summary

- How temporal models can be used to describe systems that evolve in time.
- How temporal logics (CTL*, etc.) can be used to specify those systems.
- How to use model checking in practice.
- How to relate a concrete temporal model to an abstract temporal model with simulation.
- How to implement model-checking for CTL, and counterexample generation for ACTL.