UNIVERSITY OF CAMBRIDGE

Hoare Logic and Model Checking

Conrad Watt

Computer Laboratory, University of Cambridge, UK
http://www.cl.cam.ac.uk/~caw77

CST Part II – 2018/19

Slides due to Alan Mycroft, heavily based on Mike Gordon's 2014/15 courses

Part 2: Temporal Logic and Model Checking

Big-picture view of second half of the course

- idea of model checking
- the models (Kripke structures), and getting them from real systems
- the formulae (temporal logics), expressing ideas in them and comparing them
- model abstraction

Dominic Mulligan's 2016/17 course to the same syllabus covers the same topics in a somewhat different way, and includes a lecture on practical use of the NuSMV model-checking tool.

- `http://www.cl.cam.ac.uk/teaching/1617/HLog+ModC`
- `http://nusmv.fbk.eu/`

A motivating example

```
bool flag[2] = {false, false};     int turn;

Thread 1: flag[0] = true;
          turn = 1;
          while (flag[1] && turn == 1); // busy wait
          // critical section
          flag[0] = false;
          // non-critical stuff
          repeat;

Thread 2: flag[1] = true;
          turn = 0;
          while (flag[0] && turn == 0); // busy wait
          // critical section
          flag[1] = false;
          // non-critical stuff
          repeat;
```

How can we prove this implements mutual exclusion without using locks (Peterson's algorithm)? Answer: model checking.

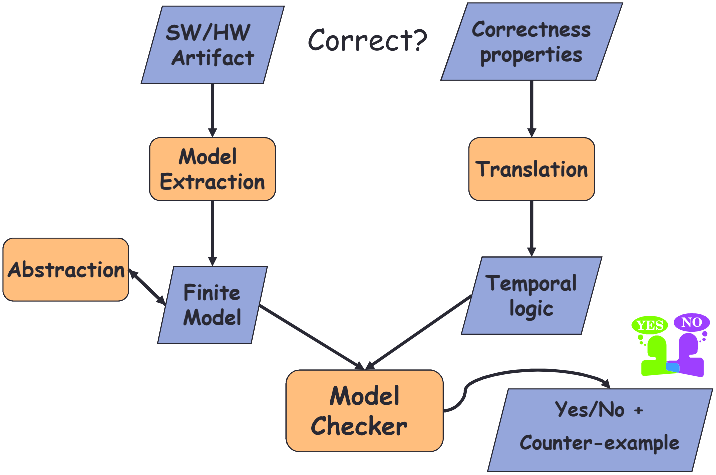
Hoare Logic vs Model Checking

Couldn't we use Hoare logic to prove it too? Perhaps (if we knew how to deal with concurrency!). Sometimes Hoare logic is easier, sometimes model checking.

- Hoare logic is built on *proof theory*, syntactically showing various formulae hold at each point in the program. Emphasis on *proof*, hence using inference rules R as we've seen to establish $\vdash_R \phi$.
- Model checking instead is built on *model theory*, exhaustive checking. E.g. we can prove a formula ϕ is valid or satisfiable by determining its *value* $\models_I \phi$ at every interpretation I of its free variables.

Very different techniques: Hoare-like logics are in principle more general, but automation is hard, and some primitives hard (e.g. concurrency). Model checking is automatic, but requires some form of finiteness in the problem for exhaustively enumerating states.

Model-checking overview



[Acknowledgement: image due to Arie Gurfinkel]

Model checking in words

Model checking is used to verify properties of a potentially complex) *hardware or software system*:

- we express the desired property, or aspects of it, as a *a modal formula* – here propositional logic augmented with *temporal operators* (e.g. **F**,**G**,**X**).
- we generally don't verify the actual system (unlike Hoare logic), but instead create a *a finite model of such a system*, and verify that. The model is expressed as *a Kripke structure* containing states and transitions, and where each state is labelled with a set of *atomic properties*.
- *a model-checking algorithm or tool* then attempts to determine the validity of the specification w.r.t. the model and either says "yes" or provides *a counter-example* trace.
- sometimes abstraction (as in 'abstract interpretation' from the Optimising Compilers course) is useful for helping us get a finite model)

Revision

[1A Digital Electronics and 1B Logic and Proof]

- Are $AB + A\overline{C} + BC$ and $BC + A\overline{C}$ equivalent?
- In other words, letting ϕ be the formula
  $$(A \wedge B) \vee (A \wedge \neg C) \vee (B \wedge C) \quad \Leftrightarrow \quad (B \wedge C) \vee (A \wedge \neg C)$$
  does $\models \phi$ hold (in propositional logic)?
- Two methods:
  - we could show $\models_I \phi$ for every *interpretation* I
  - we could prove $\vdash_R \phi$ for some set of sound and complete set of rules R (e.g. algebraic equalities like $A \vee (\mathrm{A} \wedge B) = A$)
- So far in the course (Hoare logic) we've used $\vdash$. But for propositional logic (e.g. modelling hardware) it's easier and faster to check that $\models_I \phi$ holds in all interpretations. Why? Finiteness.
  (Note that Karnaugh maps can speed up checking this.)
- Additional benefit: counter-example if something isn't true.

Revision (2)

- An *interpretation* for propositional logic with propositional variables P (say $\{A, B, C\}$) is a finite map from $\{A, B, C\}$ to $\{\mathit{true}, \mathit{false}\}$, or equivalently, the subset of $\{A, B, C\}$ which maps to *true*.
- When does a formula ϕ satisfy an interpretation I? Defined by structural induction on ϕ:
- $\models_I P$ if $P \in I$
  $\models_I \neg\phi$ if $\models_I \phi$ is false
  $\models_I \phi \wedge \phi'$ if $\models_I \phi$ and $\models_I \phi'$
- Recall that an interpretation I which makes formula ϕ true is called a *model* of ϕ. (That's why we're doing 'model checking' – determining whether a proposed model is actually one.)
  So we'll write M from now on, rather than I, for interpretations we hope are models.

Revision (3)

- Sometimes we write $[\![\phi]\!]_M$ for this (only an incidental connection to denotational semantics). So the above can alternatively be written:
  $[\![P]\!]_M = M(P)$ (treating M as a mapping here)
  $[\![\neg\phi]\!]_M = \text{not } [\![\phi]\!]_M$
  $[\![\phi \wedge \phi']\!]_M = [\![\phi]\!]_M \text{ and } [\![\phi']\!]_M$

Observation (not mentioned in Logic and Proof):

- The definition of model satisfaction $\models_I \phi$ directly gives an algorithm ($O(n)$ in the size of ϕ).

Logic and notation used in this course

- In this course we write $M \models \phi$ (and sometimes $[\![\phi]\!]_M$) rather than the $\Gamma \models_M \phi$ of Logic and Proof.
- In this course we're mainly interested in whether a formula ϕ holds in some particular putative model M, not in all interpretations. If so we say that "model M satisfies ϕ".
- We're also interested in richer formulae than propositional logic, as want to model formulae whose truth might vary over time (hence the name "temporal logic").
- We're also interested in richer models than "which propositional variables are true", so we use Kripke structures as models; these reflect systems that change state over time.

Temporal Logic and Model Checking

- Model
  - mathematical structure extracted from hardware or software; here a *Kripke structure*
- Temporal logic
  - provides a language for specifying functional properties; here a *temporal logic* (LTL or CTL, see later)
- Model checking
  - checks whether a given property holds of a model

- Model checking is a kind of static verification
  - dynamic verification is simulation (HW) or testing (SW)

A Kripke structure

We assume given a set of *atomic properties* AP.

A *(finite) Kripke structure* is a 4-tuple (S, S_0, R, L) where S is a finite set of *states*, $S_0 \subseteq S$ is the subset of possible *initial states*, R is a binary relation on states (the *transition relation*) and L is a *labelling function* mapping from S to $\mathcal{P}(AP)$.

Notes

- we often call a Kripke structure a *Kripke model*
- some authors omit S_0 and only give a 3-tuple (wrong!)
- some authors use *world* instead of state and *accessibility relation* instead of transition relation.
- note that $L(s)$ specifies a propositional model for each state $s \in S$, hence the phrase *possible worlds*.
- some authors write 2^{AP} instead of $\mathcal{P}(AP)$.

Comparison to similar structures

Computer hardware as a state machine:

- instead of R we have a transition function $next : Inp \times S \to S$ (where Inp is an input alphabet) and an output function $output : Inp \times S \to \mathcal{P}(AP)$ (viewing AP as externally visible outputs)

Finite-state automata

- instead of R we have a ternary transition relation – a subset of $\Sigma \times S \times S$ – where Σ is an alphabet).
- By having $accept \in AP$, we can recover ‘accepting states’ s as the requirement $accept \in L(s)$.

Kripke models don’t have input – they treat user-input as non-determinism. (But Part II course “Topics in Concurrency” uses richer models with an alphabet like Σ above, and a richer transition relation.)

Transition systems

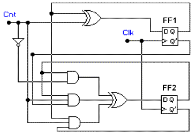
- Start by looking at the (S, R) components of a Kripke model, this is also called a transition system
  - S is a set of *states*
  - R is a *transition relation*
  - we could add start states S_0 too, but doesn't add much.
- $(s, s') \in R$ means s' can be reached from s in one step. But this notation is awkward, so:
  - here we mainly write $R\ s\ s$; treating relation R as being the equivalent function $R : S \to (S \to \mathbb{B})$ (where $\mathbb{B} = \{true, false\}$)
  - i.e. $R_{\text{(this course)}}\ s\ s' \;\Leftrightarrow\; (s, s') \in R_{\text{(formally)}}$
  - some books also write $R(s, s')$ (equivalent by currying)
- we'll consider AP later.

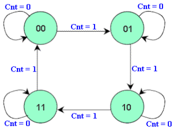
A simple example transition system

- A simple T.S.: $(\underbrace{\{0,1,2,3\}}_{S}, \underbrace{\lambda n\ n'.\ n' = n{+}1 (mod\ 4)}_{R})$
  - where "$\lambda x.\ \cdots x \cdots$" is the function mapping x to $\cdots x \cdots$
  - so $R\ n\ n' = (n' = n{+}1 (mod\ 4))$
  - e.g. $R\ 0\ 1 \wedge R\ 1\ 2 \wedge R\ 2\ 3 \wedge R\ 3\ 0$

$$\begin{tikzcd}
0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[lll, bend left]
\end{tikzcd}$$

- Might be extracted from:





[Acknowledgement: http://eelab.usyd.edu.au/digital_tutorial/part3/t-diag.htm]

DIV: a software example

- Perhaps a familiar program:

```
0:   R:=X;
1:   Q:=0;
2:   WHILE Y≤R DO
3:   (R:=R-Y;
4:    Q:=Q+1)
5:
```

- State (pc, x, y, r, q)
  - $pc \in \{0, 1, 2, 3, 4, 5\}$ program counter
  - $x, y, r, q \in \mathbb{Z}$ are the values of X, Y, R, Q
- Model $(S_{\mathtt{DIV}}, R_{\mathtt{DIV}})$ where:

$$S_{\mathtt{DIV}} = [0..5] \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \quad \text{(where } [m..n] = \{m, m{+}1, \ldots, n\}\text{)}$$

$$\begin{array}{llr}
\forall x\ y\ r\ q.\ R_{\mathtt{DIV}}\ (0, x, y, r, q)\ (1, x, y, x, q) & & \wedge \\
\quad R_{\mathtt{DIV}}\ (1, x, y, r, q)\ (2, x, y, r, 0) & & \wedge \\
\quad R_{\mathtt{DIV}}\ (2, x, y, r, q)\ ((\mathit{if}\ y{\le}r\ \mathit{then}\ 3\ \mathit{else}\ 5), x, y, r, q) & & \wedge \\
\quad R_{\mathtt{DIV}}\ (3, x, y, r, q)\ (4, x, y, (r{-}y), q) & & \wedge \\
\quad R_{\mathtt{DIV}}\ (4, x, y, r, q)\ (2, x, y, r, (q{+}1)) & &
\end{array}$$

Deriving a transition system from a state machine

- State machine transition function: $\delta \ : \ \mathit{Inp} \times \mathit{Mem} \to \mathit{Mem}$
  - Inp is a set of inputs
  - Mem is a memory (set of storable values)
- Transition system is: (S_δ, R_δ) where:

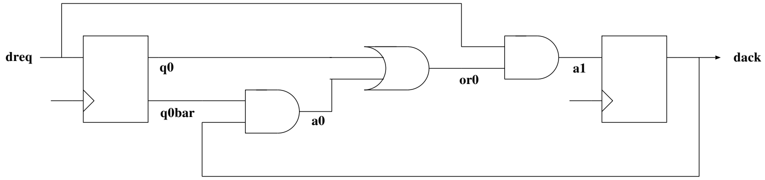
  $S_\delta = \mathit{Inp} \times \mathit{Mem}$

  $R_\delta \, (i, m) \, (i', m') \ = \ (m' \ = \ \delta(i, m))$

  and
  - i' arbitrary: determined by environment not by machine
  - m' determined by input and current state of machine
- Deterministic machine, non-deterministic transition relation
  - inputs unspecified (determined by environment)
  - so called “input non-determinism”

RCV: example state-machine circuit specification

- Part of a handshake circuit:



- Input: $dreq$, Memory: $(q0, dack)$
- Relationships between Boolean values on wires:

$$\begin{array}{lcl} q0bar & = & \neg q0 \\ a0 & = & q0bar \wedge dack \\ or0 & = & q0 \vee a0 \\ a1 & = & dreq \wedge or0 \end{array}$$

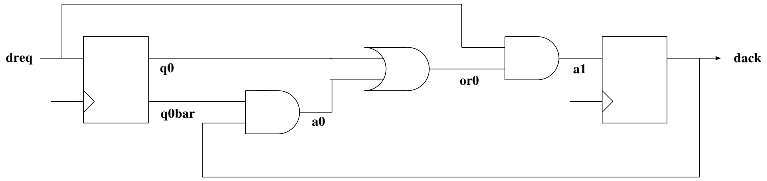
- State machine: $\delta_{\mathrm{RCV}} : \mathbb{B} \times (\mathbb{B}\times\mathbb{B})\to(\mathbb{B}\times\mathbb{B})$

$$\delta_{\mathrm{RCV}}\ (\underbrace{dreq,}_{Inp}\ \underbrace{(q0, dack)}_{Mem}) = (dreq,\ dreq \wedge (q0 \vee (\neg q0 \wedge dack)))$$

- RTL model – could have lower level model with clock edges

RCV: deriving a transition system

- Circuit from previous slide:



- State represented by a triple of Booleans $(dreq, q0, dack)$
- By De Morgan Law: $q0 \lor (\neg q0 \land dack) = q0 \lor dack$
- Hence δ_{RCV} corresponds to transition system $(S_{\mathrm{RCV}}, R_{\mathrm{RCV}})$ where:

  $S_{\mathrm{RCV}} = \mathbb{B} \times \mathbb{B} \times \mathbb{B}$ [identifying $\mathbb{B} \times \mathbb{B} \times \mathbb{B}$ with $\mathbb{B} \times (\mathbb{B} \times \mathbb{B})$]

  $$R_{\mathrm{RCV}}\ (dreq, q0, dack)\ (dreq', q0', dack') = (q0' = dreq) \land (dack' = (dreq \land (q0 \lor dack)))$$

- but drawing R pictorially can be clearer ...

RCV as a transition system

- Possible states for RCV:
  $\{000, 001, 010, 011, 100, 101, 110, 111\}$
  where $b_2 b_1 b_0$ denotes state
  $\texttt{dreq} = b_2 \wedge \texttt{q0} = b_1 \wedge \texttt{dack} = b_0$
- Graph of the transition relation:

$$\begin{tikzcd}
000 \arrow[loop above] \arrow[rr] & & 100 \arrow[r] \arrow[dd, shift left] & 110 \arrow[r] \arrow[dddr] & 111 \arrow[loop above] \arrow[ddd] & \\
& 001 \arrow[ul] \arrow[ur] & & & & 101 \arrow[ul] \arrow[ddl] \\
& & 010 \arrow[uu, shift left] \arrow[uull] & & & \\
& & & & 011 \arrow[uuull] \arrow[uuullll] &
\end{tikzcd}$$

Some comments

- $R_{\mathtt{RCV}}$ is non-deterministic and left-total
  - $R_{\mathtt{RCV}}\ (1,1,1)\ (0,1,1)$ and $R_{\mathtt{RCV}}\ (1,1,1)\ (1,1,1)$ (where $1 = \mathit{true}$ and $0 = \mathit{false}$)
  - $R_{\mathtt{RCV}}\ (\mathit{dreq}, q0, \mathit{dack})\ (\mathit{dreq}', \mathit{dreq}, (\mathit{dreq} \wedge (q0 \vee \mathit{dack})))$
- $R_{\mathtt{DIV}}$ is deterministic but not left-total
  - at most one successor state
  - no successor when $\mathit{pc} = 5$
- Non-deterministic models are very common, e.g. from:
  - asynchronous hardware
  - parallel software (more than one thread)
- Can extend any transition relation R to be left-total, e.g.
  $R^{\text{total}} = R \cup \{(s,s) \mid \neg\exists s' \text{ such that } (s,s') \in R\}$
  - some texts require left-totality (e.g. *Model Checking* by Clarke et al.); this can simplify reasoning.

`JM1`: a non-deterministic software example

- From Jhala and Majumdar's tutorial:

```
Thread 1                          Thread 2
0:   IF LOCK=0 THEN LOCK:=1;      0:   IF LOCK=0 THEN LOCK:=1;
1:   X:=1;                        1:   X:=2;
2:   IF LOCK=1 THEN LOCK:=0;      2:   IF LOCK=1 THEN LOCK:=0;
3:                                3:
```

- Two program counters, state: $(pc_1, pc_2, lock, x)$

$$S_{\mathrm{JM1}} = [0..3] \times [0..3] \times \mathbb{Z} \times \mathbb{Z}$$

$$\begin{array}{lllll}
\forall pc_1\ pc_2\ lock\ x. & R_{\mathrm{JM1}}\ (0, pc_2, 0, x) & (1, pc_2, 1, x) & \wedge \\
& R_{\mathrm{JM1}}\ (1, pc_2, lock, x) & (2, pc_2, lock, 1) & \wedge \\
& R_{\mathrm{JM1}}\ (2, pc_2, 1, x) & (3, pc_2, 0, x) & \wedge \\
& R_{\mathrm{JM1}}\ (pc_1, 0, 0, x) & (pc_1, 1, 1, x) & \wedge \\
& R_{\mathrm{JM1}}\ (pc_1, 1, lock, x) & (pc_1, 2, lock, 2) & \wedge \\
& R_{\mathrm{JM1}}\ (pc_1, 2, 1, x) & (pc_1, 3, 0, x) &
\end{array}$$

- Non-deterministic:

$$R_{\mathrm{JM1}}\ (0, 0, 0, x)\ (1, 0, 1, x)$$
$$R_{\mathrm{JM1}}\ (0, 0, 0, x)\ (0, 1, 1, x)$$

- Not so obvious that R_{JM1} is a correct model

Atomic properties (properties of states)

- Atomic properties are true or false of individual *states*
  - an atomic property p is a function $p : S \to \mathbb{B}$
  - can also be regarded as a subset of state: $p \subseteq S$
- Example atomic properties of `RCV` (where $1 = \mathit{true}$ and $0 = \mathit{false}$)

$$\begin{aligned}
\texttt{Dreq}(dreq, q0, dack) &= (dreq = 1)\\
\texttt{NotQ0}(dreq, q0, dack) &= (q0 = 0)\\
\texttt{Dack}(dreq, q0, dack) &= (dack = 1)\\
\texttt{NotDreqAndQ0}(dreq, q0, dack) &= (dreq{=}0) \wedge (q0{=}1)
\end{aligned}$$

- Example atomic properties of `DIV`

$$\begin{aligned}
\texttt{AtStart}\,(pc, x, y, r, q) &= (pc = 0)\\
\texttt{AtEnd}\,(pc, x, y, r, q) &= (pc = 5)\\
\texttt{InLoop}\,(pc, x, y, r, q) &= (pc \in \{3, 4\})\\
\texttt{YleqR}\,(pc, x, y, r, q) &= (y \leq r)\\
\texttt{Invariant}\,(pc, x, y, r, q) &= (x = r + (y \times q))
\end{aligned}$$

Atomic properties as labellings

These properties are convenient to express:

$$\begin{aligned}
\texttt{Dreq}(dreq, q0, dack) &= (dreq = 1)\\
\texttt{NotQ0}(dreq, q0, dack) &= (q0 = 0)\\
\texttt{Dack}(dreq, q0, dack) &= (dack = 1)\\
\texttt{NotDreqAndQ0}(dreq, q0, dack) &= (dreq{=}0) \wedge (q0{=}1)
\end{aligned}$$

But how are they related to the Kripke model requirement at "each state is labelled with a set of atomic properties"?

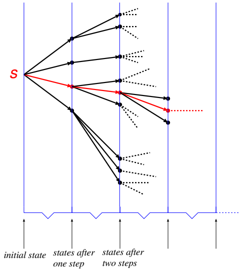
These are just equivalent views. Note that states $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(1,1,1)$ are labelled with $\texttt{Dreq} \in AP$, and no other state is. Similarly for $\texttt{NotQ0}, \texttt{Dack}, \texttt{NotDreqAndQ0}$.

So the labelling function $L : S \to \mathcal{P}(AP)$ is here given by

$$\begin{aligned}
L(0,0,0) &= \{\texttt{NotQ0}\}\\
L(1,0,0) &= \{\texttt{Dreq}, \texttt{NotQ0}\}\\
\text{etc}
\end{aligned}$$

Model behaviour viewed as a computation tree

- Atomic properties are true or false of individual states
- General properties are true or false of whole behaviour
- Behaviour of (S, R) starting from $s \in S$ as a tree:



- A path is shown in red
- Properties may look at all paths, or just a single path
  - CTL: Computation Tree Logic (all paths from a state)
  - LTL: Linear Temporal Logic (a single path)

Paths

- A path of (S, R) is represented by a function $\pi : \mathbb{N} \to S$
  - $\pi(i)$ is the ith element of π (first element is $\pi(0)$)
  - might sometimes write $\pi\ i$ instead of $\pi(i)$
  - $\pi{\downarrow}i$ is the i-th tail of π so $\pi{\downarrow}i(n) = \pi(i + n)$
  - successive states in a path must be related by R

- Path $R\ s\ \pi$ is true if and only if π is a path starting at s:

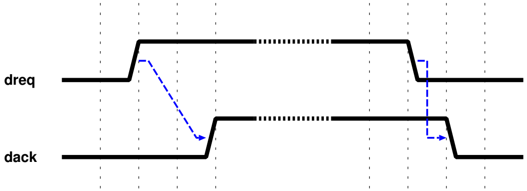
  $$\text{Path}\ R\ s\ \pi\ =\ (\pi(0) = s)\ \wedge\ \forall i.\ R\ (\pi(i))\ (\pi(i{+}1))$$

  where:

  $$\text{Path} : \underbrace{(S \to S \to \mathbb{B})}_{\text{transition relation}} \to \underbrace{S}_{\text{initial state}} \to \underbrace{(\mathbb{N} \to S)}_{\text{path}} \to \mathbb{B}$$

RCV: example hardware properties

- Consider this timing diagram:



- Two handshake properties representing the diagram:
  - following a rising edge on dreq, the value of dreq remains 1 (i.e. *true*) until it is acknowledged by a rising edge on dack
  - following a falling edge on dreq, the value on dreq remains 0 (i.e. *false*) until the value of dack is 0
- A property language is used to formalise such properties. In this course this is some form of temporal logic.

DIV: example program properties

```
0:   R:=X;
1:   Q:=0;
2:   WHILE Y≤R DO
3:   (R:=R-Y;
4:    Q:=Q+1)
5:
```

$$
\begin{aligned}
\texttt{AtStart}\,(pc, x, y, r, q) &= (pc = 0)\\
\texttt{AtEnd}\,(pc, x, y, r, q) &= (pc = 5)\\
\texttt{InLoop}\,(pc, x, y, r, q) &= (pc \in \{3, 4\})\\
\texttt{YleqR}\,(pc, x, y, r, q) &= (y \le r)\\
\texttt{Invariant}\,(pc, x, y, r, q) &= (x = r + (y \times q))
\end{aligned}
$$

- Example properties of the program DIV.
  - on every execution if AtEnd is true then Invariant is true and YleqR is not true
  - on every execution there is a state where AtEnd is true
  - on any execution if there exists a state where YleqR is true then there is also a state where InLoop is true
- Compare these with what is expressible in Hoare logic
  - execution: a path starting from a state satisfying AtStart

Recall `JM1`: a non-deterministic program example

```
Thread 1                              Thread 2
0:   IF LOCK=0 THEN LOCK:=1;          0:   IF LOCK=0 THEN LOCK:=1;
1:   X:=1;                            1:   X:=2;
2:   IF LOCK=1 THEN LOCK:=0;          2:   IF LOCK=1 THEN LOCK:=0;
3:                                    3:
```

$$S_{\mathtt{JM1}} = [0..3] \times [0..3] \times \mathbb{Z} \times \mathbb{Z}$$

$$\begin{array}{lllll}
\forall pc_1\ pc_2\ lock\ x.\ & R_{\mathtt{JM1}}\ (0, pc_2, 0, x) & (1, pc_2, 1, x) & \wedge \\
& R_{\mathtt{JM1}}\ (1, pc_2, lock, x) & (2, pc_2, lock, 1) & \wedge \\
& R_{\mathtt{JM1}}\ (2, pc_2, 1, x) & (3, pc_2, 0, x) & \wedge \\
& R_{\mathtt{JM1}}\ (pc_1, 0, 0, x) & (pc_1, 1, 1, x) & \wedge \\
& R_{\mathtt{JM1}}\ (pc_1, 1, lock, x) & (pc_1, 2, lock, 2) & \wedge \\
& R_{\mathtt{JM1}}\ (pc_1, 2, 1, x) & (pc_1, 3, 0, x) &
\end{array}$$

- An atomic property:
  - `NotAt11`$(pc_1, pc_2, lock, x) = \neg((pc_1 = 1) \wedge (pc_2 = 1))$
- A non-atomic property:
  - all states reachable from $(0, 0, 0, 0)$ satisfy `NotAt11`
  - this is an example of a reachability property

All states reachable from $(0,0,0,0)$ satisfy `NotAt11`

```
Thread 1                           Thread 2
0:  IF LOCK=0 THEN LOCK:=1;        0:  IF LOCK=0 THEN LOCK:=1;
1:  X:=1;                          1:  X:=2;
2:  IF LOCK=1 THEN LOCK:=0;        2:  IF LOCK=1 THEN LOCK:=0;
3:                                 3:
```

$$\begin{array}{ll|ll} R_{\mathrm{JM1}}\ (0,pc_2,0,x) & (1,pc_2,1,x) & R_{\mathrm{JM1}}\ (pc_1,0,0,x) & (pc_1,1,1,x) \\ R_{\mathrm{JM1}}\ (1,pc_2,lock,x) & (2,pc_2,lock,1) & R_{\mathrm{JM1}}\ (pc_1,1,lock,x) & (pc_1,2,lock,2) \\ R_{\mathrm{JM1}}\ (2,pc_2,1,x) & (3,pc_2,0,x) & R_{\mathrm{JM1}}\ (pc_1,2,1,x) & (pc_1,3,0,x) \end{array}$$

- `NotAt11`$(pc_1, pc_2, lock, x) = \neg((pc_1 = 1) \wedge (pc_2 = 1))$
- Can only reach $pc_1 = 1 \ \wedge \ pc_2 = 1$ via:

$$\begin{array}{llll} R_{\mathrm{JM1}}\ (0,pc_2,0,x) & (1,pc_2,1,x) & \text{i.e. a step } R_{\mathrm{JM1}}\ (0,1,0,x) & (1,1,1,x) \\ R_{\mathrm{JM1}}\ (pc_1,0,0,x) & (pc_1,1,1,x) & \text{i.e. a step } R_{\mathrm{JM1}}\ (1,0,0,x) & (1,1,1,x) \end{array}$$

- But:

$$\begin{array}{l} R_{\mathrm{JM1}}\ (pc_1,pc_2,lock,x)\ (pc_1',pc_2',lock',x') \ \wedge \ pc_1'{=}0 \ \wedge \ pc_2'{=}1 \ \Rightarrow \ lock'{=}1 \\ \wedge \\ R_{\mathrm{JM1}}\ (pc_1,pc_2,lock,x)\ (pc_1',pc_2',lock',x') \ \wedge \ pc_1'{=}1 \ \wedge \ pc_2'{=}0 \ \Rightarrow \ lock'{=}1 \end{array}$$

- So can never reach $(0,1,0,x)$ or $(1,0,0,x)$
- So can't reach $(1,1,1,x)$, hence never $(pc_1 = 1) \wedge (pc_2 = 1)$
- Hence all states reachable from $(0,0,0,0)$ satisfy `NotAt11`

Reachability

- $R\ s\ s'$ means s' reachable from s in one step
- $R^n\ s\ s'$ means s' reachable from s in n steps

  $$\begin{array}{lcl} R^0\ s\ s' & = & (s = s') \\ R^{n+1}\ s\ s' & = & \exists s''.\ R\ s\ s'' \wedge R^n\ s''\ s' \end{array}$$
- $R^*\ s\ s'$ means s' reachable from s in finite steps

  $R^*\ s\ s' = \exists n.\ R^n\ s\ s'$
- Note: $R^*\ s\ s' \Leftrightarrow \exists \pi\ n.\ \text{Path}\ R\ s\ \pi \wedge (s' = \pi(n))$
- The set of states reachable from s is $\{s' \mid R^*\ s\ s'\}$
- Verification problem: all states reachable from s satisfy p
  - verify truth of $\forall s'.\ R^*\ s\ s' \Rightarrow p(s')$
  - e.g. all states reachable from $(0,0,0,0)$ satisfy $\texttt{NotAt11}$
  - i.e. $\forall s'.\ R^*_{\texttt{JM1}}\ (0,0,0,0)\ s' \Rightarrow \texttt{NotAt11}(s')$

Model Checking a Simple Property

Models and model checking

- We've defined and exemplified Kripke models
- We treat their states as externally unimportant, what is important is how the various atomic predicates change as the Kripke model evolves.
- A *Kripke structure* is a tuple (S, S_0, R, L) where L is a labelling function from S to $\mathcal{P}(AP)$
  - Note the two understandings of atomic properties:
  - the formal one above $p \in AP$
  - the previous informal, but equivalent, one $\lambda s.\ p \in L(s)$
  - often convenient to assume $\mathtt{T}, \mathtt{F} \in AP$ with $\forall s$: $\mathtt{T} \in L(s)$ and $\mathtt{F} \notin L(s)$
- Model checking computes whether $(S, S_0, R, L) \models \phi$
  - ϕ is a property expressed in a property language
  - informally $M \models \phi$ means "formula ϕ is true in model M"

Start with trivial and minimal property languages . . .

Trivial property language: ϕ is p where $p \in AP$

- Assume $M = (S, S_0, R, L)$
- $M \models p$ means p true of all initial states of M
- formally $M \models p$ holds if $\forall s \in S_0.\ p \in L(s)$
- uninteresting – does not consider transitions in M (other 'possible worlds' than the initial ones)

Minimal property language: ϕ is $\mathbf{AG}\,p$ where $p \in AP$

Our first temporal operator in a very restricted form so far.

- Consider properties ϕ of form $\mathbf{AG}\,p$ where $p \in AP$
  - "**AG**" stands for "Always Globally"
  - from CTL (same meaning, more elaborately expressed)

- Assume $M = (S, S_0, R, L)$

- Reachable states of M are $\{s' \mid \exists s \in S_0.\ R^*\ s\ s'\}$
  - i.e. the set of states reachable from an initial state

- Define $\text{Reachable}\ M = \{s' \mid \exists s \in S_0.\ R^*\ s\ s'\}$

- $M \models \mathbf{AG}\,p$ means p true of all reachable states of M

- If $M = (S, S_0, R, L)$ then $M \models \phi$ formally defined by:

$$\boxed{M \models \mathbf{AG}\,p \Leftrightarrow \forall s'.\ s' \in \text{Reachable}\ M \Rightarrow p \in L(s')}$$

Model checking $M \models \mathbf{AG}\, p$

- $M \models \mathbf{AG}\, p \Leftrightarrow \forall s'.\ s' \in \text{Reachable } M \Rightarrow p \in L(s')$
  $\Leftrightarrow \text{Reachable } M \subseteq \{s' \mid p \in L(s')\}$

  checked by:
  - first computing Reachable M
  - then checking p true of all its members
- Let $\mathcal{S}$ abbreviate $\{s' \mid \exists s \in S_0.\ R^*\ s\ s'\}$ (i.e. Reachable M)
- Compute $\mathcal{S}$ iteratively: $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \cdots \cup \mathcal{S}_n \cup \cdots$
  - i.e. $\mathcal{S} = \bigcup_{n=0}^{\infty} \mathcal{S}_n$
  - where: $\mathcal{S}_0 = S_0$ (set of initial states)
  - and inductively: $\mathcal{S}_{n+1} = \mathcal{S}_n \cup \{s' \mid \exists s \in \mathcal{S}_n \wedge R\ s\ s'\}$
- Clearly $\mathcal{S}_0 \subseteq \mathcal{S}_1 \subseteq \cdots \subseteq \mathcal{S}_n \subseteq \cdots$
- Hence if $\mathcal{S}_{\mathrm{m}} = \mathcal{S}_{\mathrm{m}+1}$ then $\mathcal{S} = \mathcal{S}_{\mathrm{m}}$
- Algorithm: compute $\mathcal{S}_0, \mathcal{S}_1, \ldots$, until no change;
  then check p labels all members of computed set

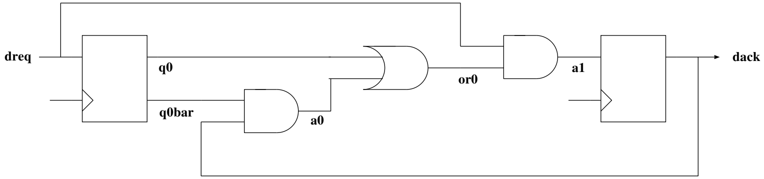
Algorithmic issues

Compute $\mathcal{S}_0$, $\mathcal{S}_1$, $\ldots$, until no change;
then check p holds of all members of computed set

- Does the algorithm terminate?
  - yes, if set of states is finite, because then no infinite chains:
    $\mathcal{S}_0 \subset \mathcal{S}_1 \subset \cdots \subset \mathcal{S}_n \subset \cdots$
- How to represent $\mathcal{S}_0$, $\mathcal{S}_1$, $\ldots$?
  - explicitly (e.g. lists or something more clever)
  - symbolic expression
- Huge literature on calculating set of reachable states

Example: RCV

- Recall the handshake circuit:



- State represented by a triple of Booleans $(dreq, q0, dack)$
- A model of RCV is M_{RCV} where:

  $M = (S_{\mathrm{RCV}}, \{(1,1,1)\}, R_{\mathrm{RCV}}, L_{\mathrm{RCV}})$

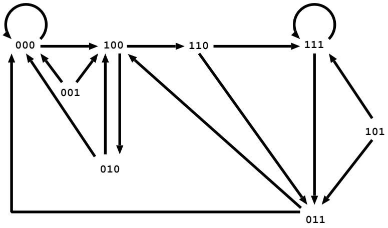
  and

  $$\begin{aligned} R_{\mathrm{RCV}}\ & (dreq, q0, dack)\ (dreq', q0', dack') = \\ & (q0' = dreq) \land (dack' = (dreq \land (q0 \lor dack))) \end{aligned}$$

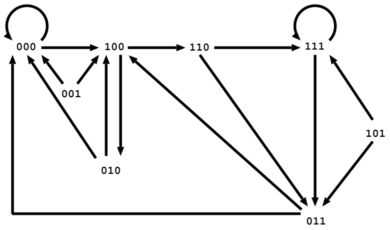
- AP and labelling function L_{RCV} discussed later

RCV as a transition system

- Possible states for RCV:
  $\{000, 001, 010, 011, 100, 101, 110, 111\}$
  where $b_2 b_1 b_0$ denotes state
  $\texttt{dreq} = b_2 \land \texttt{q0} = b_1 \land \texttt{dack} = b_0$
- Graph of the transition relation:



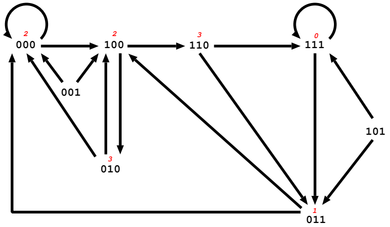
Computing Reachable M_{RCV}



- Define:

$$
\begin{aligned}
\mathcal{S}_0 &= \{b_2 b_1 b_0 \mid b_2 b_1 b_0 \in \{111\}\} \\
&= \{111\} \\
\mathcal{S}_{i+1} &= \mathcal{S}_i \;\cup\; \{s' \mid \exists s \in \mathcal{S}_i.\; R_{\mathrm{RCV}}\; s\; s'\} \\
&= \mathcal{S}_i \;\cup\; \{b_2' b_1' b_0' \mid \\
&\qquad\quad \exists b_2 b_1 b_0 \in \mathcal{S}_i.\; (b_1' = b_2) \;\wedge\; (b_0' = b_2 \wedge (b_1 \vee b_0))\}
\end{aligned}
$$

Computing Reachable M_{RCV} (continued)



- Compute:

$$
\begin{aligned}
\mathcal{S}_0 &= \{111\} \\
\mathcal{S}_1 &= \{111\} \ \cup \ \{011\} \\
&= \{111, 011\} \\
\mathcal{S}_2 &= \{111, 011\} \ \cup \ \{000, 100\} \\
&= \{111, 011, 000, 100\} \\
\mathcal{S}_3 &= \{111, 011, 000, 100\} \ \cup \ \{010, 110\} \\
&= \{111, 011, 000, 100, 010, 110\} \\
\mathcal{S}_i &= \mathcal{S}_3 \quad (i > 3)
\end{aligned}
$$

- Hence Reachable $M_{\mathrm{RCV}} = \{111, 011, 000, 100, 010, 110\}$

Model checking $M_{\mathrm{RCV}} \models \mathbf{AG}\, p$

- $M = (S_{\mathrm{RCV}}, \{111\}, R_{\mathrm{RCV}}, L_{\mathrm{RCV}})$
- To check $M_{\mathrm{RCV}} \models \mathbf{AG}\, p$
  - compute $\mathrm{Reachable}\ M_{\mathrm{RCV}} = \{111, 011, 000, 100, 010, 110\}$
  - check $\mathrm{Reachable}\ M_{\mathrm{RCV}} \subseteq \{s \mid p \in L_{\mathrm{RCV}}(s)\}$
  - i.e. check if $s \in \mathrm{Reachable}\ M_{\mathrm{RCV}}$ then $p \in L_{\mathrm{RCV}}(s)$, i.e.:

    $$\begin{array}{l} p \in L_{\mathrm{RCV}}(111) \land \\ p \in L_{\mathrm{RCV}}(011) \land \\ p \in L_{\mathrm{RCV}}(000) \land \\ p \in L_{\mathrm{RCV}}(100) \land \\ p \in L_{\mathrm{RCV}}(010) \land \\ p \in L_{\mathrm{RCV}}(110) \end{array}$$
- Example
  - if $AP = \{\mathtt{A}, \mathtt{B}\}$
  - and $L_{\mathrm{RCV}}(s) \ = \ \mathit{if}\ s \in \{001, 101\}\ \mathit{then}\ \{\mathtt{A}\}\ \mathit{else}\ \{\mathtt{B}\}$
  - then $M_{\mathrm{RCV}} \models \mathbf{AG}\, \mathtt{A}$ is not true, but $M_{\mathrm{RCV}} \models \mathbf{AG}\, \mathtt{B}$ is true

Generating counterexamples (explicit states)

Can use ‘failure to prove’ to generate counter-example traces.
Here ‘trace’ is a synonym for ‘path’
Easy, but potentially slow using explicit state representation:

- Suppose not all reachable states of model M satisfy p
- i.e. $\exists s \in \text{Reachable } M.\ \neg p(s)$
- Set of reachable states $\mathcal{S}$ given by: $\mathcal{S} = \bigcup_{n=0}^{\infty} \mathcal{S}_n$
- Iterate to find least n such that $\exists s \in \mathcal{S}_n.\ \neg p(s)$
  (helpful to report the the shortest error trace)
- Pick a state s_n such that $s_n \in \mathcal{S}_n \wedge \neg p(s_n)$
- Find a path $s_0 \in S_0 \ldots s_n \in \mathcal{S}_n$ to it

Finding a path in (S, S_0, R, L) is linear time if we store breadcrumbs (back-pointers from each reachable state s_{i+1} to the state s_i which first caused it to be visited); s_0 is necessarily part of $S_0 = \mathcal{S}_0$

Explicit vs Symbolic model checking

The problem:

- Suppose we have a system with n flip-flops. Then it has up to 2^n states. Exploring all these exhaustively is exponentially horrid – even a system with three 32-bit registers has 2^{96} states which take 'forever' to explore
- In general the number of states is exponential in the number of variables and number of parallel threads.

Technology to avoid this: 'Symbolic model checking'

- Same model-checking idea
- Use symbolic representations of data (e.g. BDDs) instead of explicit state and relation representations (e.g. set of tuples of booleans)
- Do this both for states and for the transition relation
- Faster (for data-structures-and-algorithms reasons)

Symbolic Boolean model checking of reachability

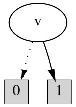
- Assume states are n-tuples of Booleans $(b_1, \ldots, b_n)$
  - $b_i \in \mathbb{B} = \{\mathit{true}, \mathit{false}\}$ $(= \{1, 0\})$
  - $S = \mathbb{B}^n$, so S is finite: 2^n states
- Assume n distinct Boolean variables: v_1,...,v_n
  - e.g. if $n = 3$ then could have $v_1 = \mathtt{x}$, $v_2 = \mathtt{y}$, $v_3 = \mathtt{z}$
- Boolean formula $f(v_1, \ldots, v_n)$ represents a subset of S
  - $f(v_1, \ldots, v_n)$ only contains variables v_1,...,v_n
  - $f(b_1, \ldots, b_n)$ denotes result of substituting b_i for v_i
  - $f(v_1, \ldots, v_n)$ determines $\{(b_1, \ldots, b_n) \mid f(b_1, \ldots, b_n) \Leftrightarrow \mathit{true}\}$
- Example $\neg(\mathtt{x} = \mathtt{y})$ represents $\{(\mathit{true}, \mathit{false}), (\mathit{false}, \mathit{true})\}$
- Transition relations also represented by Boolean formulae
  - e.g. $R_{\mathtt{RCV}}$ represented by:
    $(q0' = dreq) \wedge (dack' = (dreq \wedge (q0 \vee (\neg q0 \wedge dack))))$

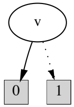
Symbolically represent Boolean formulae as BDDs

- Key features of Binary Decision Diagrams (BDDs):
  - canonical (given a variable ordering)
  - efficient to manipulate
- Variables:

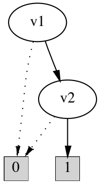
```
v    =  if v then 1 else 0
¬v   =  if v then 0 else 1
```

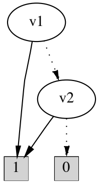
- Example: BDDs of variable `v` and `¬v`





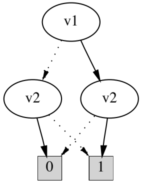
- Example: BDDs of `v1` $\land$ `v2` and `v1` $\lor$ `v2`



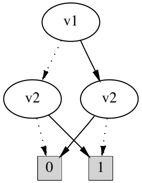


More BDD examples

- BDD of $\texttt{v1} = \texttt{v2}$



- BDD of $\texttt{v1} \neq \texttt{v2}$

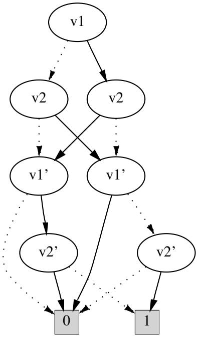


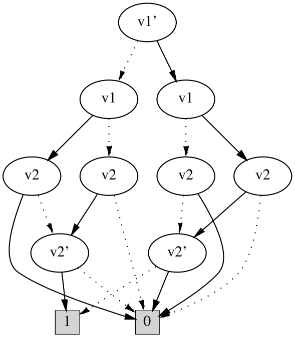
BDD of a transition relation

- BDDs of

$$(\mathtt{v1}' = (\mathtt{v1} = \mathtt{v2})) \wedge (\mathtt{v2}' = (\mathtt{v1} \neq \mathtt{v2}))$$

  with two different variable orderings





- **Exercise:** draw BDD of R_{RCV}

Standard BDD operations

- If formulae f_1, f_2 represents sets S_1, S_2, respectively then $f_1 \wedge f_2$, $f_1 \vee f_2$ represent $S_1 \cap S_2$, $S_1 \cup S_2$ respectively
- Standard algorithms compute Boolean operation on BDDs
- Abbreviate $(v_1, \ldots, v_n)$ to $\vec{v}$
- If $f(\vec{v})$ represents S
  and $g(\vec{v}, \vec{v}')$ represents $\{(\vec{v}, \vec{v}') \mid R\ \vec{v}\ \vec{v}'\}$
  then $\exists \vec{u}.\ f(\vec{u}) \wedge g(\vec{u}, \vec{v})$ represents $\{\vec{v} \mid \exists \vec{u}.\ \vec{u} \in S \wedge R\ \vec{u}\ \vec{v}\}$
- Can compute BDD of $\exists \vec{u}.\ h(\vec{u}, \vec{v})$ from BDD of $h(\vec{u}, \vec{v})$
  - e.g. BDD of $\exists v_1.\ h(v_1, v_2)$ is BDD of $h(\mathtt{T}, v_2) \vee h(\mathtt{F}, v_2)$
- From BDD of formula $f(v_1, \ldots, v_n)$ can compute b_1, ..., b_n such that if $v_1 = b_1$, ..., $v_n = b_n$ then $f(b_1, \ldots, b_n) \Leftrightarrow \mathit{true}$
  - b_1, ..., b_n is a satisfying assignment (SAT problem)
  - used for counterexample generation (see later)

Reachable States via BDDs

- Assume $M = (S, S_0, R, L)$ and $S = \mathbb{B}^n$
- Represent R by Boolean formulae $g(\vec{v}, \vec{v'})$
- Iteratively define formula $f_n(\vec{v})$ representing $\mathcal{S}_n$

$$\begin{aligned} f_0(\vec{v}) &= \text{formula representing } S_0 \\ f_{n+1}(\vec{v}) &= f_n(\vec{v}) \vee (\exists \vec{u}.\ f_n(\vec{u}) \wedge g(\vec{u}, \vec{v})) \end{aligned}$$

- Let $\mathcal{B}_0$, $\mathcal{B}_R$ be BDDs representing $f_0(\vec{v})$, $g(\vec{v}, \vec{v'})$
- Iteratively compute BDDs $\mathcal{B}_n$ representing f_n

$$\mathcal{B}_{n+1} = \mathcal{B}_n \ \underline{\vee}\ \ (\underline{\exists \vec{u}.}\ \ \mathcal{B}_n\underline{[\vec{u}/\vec{v}]}\ \underline{\wedge}\ \mathcal{B}_R\underline{[\vec{u}, \vec{v}/\vec{v}, \vec{v'}]})$$

  - efficient using (blue underlined) standard BDD algorithms (renaming, conjunction, disjunction, quantification)
  - BDD $\mathcal{B}_n$ only contains variables $\vec{v}$: represents $\mathcal{S}_n \subseteq S$
- At each iteration check $\mathcal{B}_{n+1} = \mathcal{B}_n$ efficient using BDDs
  - when $\mathcal{B}_{n+1} = \mathcal{B}_n$ can conclude $\mathcal{B}_n$ represents Reachable M
  - we call this BDD $\mathcal{B}_M$ in a later slide (i.e. $\mathcal{B}_M = \mathcal{B}_n$)

Engineering BDDs is significant work

- size of BDD can depend hugely on choice of ‘variable order’
- some operations (e.g. multiplication) produces big BDDs
- interleaved concurrency (think threads) can mean that the exact BDD for R is huge.
- But there are tricks beyond this course (e.g. ‘disjunctive partitioning’) which can calculate things like f_n above without computing R.
- See more-advanced courses e.g. `http://www.cs.ucsb.edu/~bultan/courses/267/`

Verification and counterexamples

- Typical safety question:
  - is property p true in all reachable states?
  - i.e. check $M \models \mathbf{AG}\, p$
  - i.e. is $\forall s.\ s \in \mathrm{Reachable}\ M \Rightarrow p\ s$

- Check using BDDs
  - compute BDD $\mathcal{B}_M$ of $\mathrm{Reachable}\ M$
  - compute BDD $\mathcal{B}_p$ of $p(\vec{v})$
  - check if BDD of $\mathcal{B}_M \underline{\Rightarrow} \mathcal{B}_p$ is the single node $\boxed{1}$

- Valid because *true* represented by a unique BDD (canonical property)

- If BDD is not $\boxed{1}$ can get counterexample

Generating counterexamples (general idea)

BDD algorithms can find satisfying assignments (SAT)

- Suppose not all reachable states of model M satisfy p
- i.e. $\exists s \in \text{Reachable } M.\ \neg(p(s))$
- Set of reachable state $\mathcal{S}$ given by: $\mathcal{S} = \bigcup_{n=0}^{\infty} \mathcal{S}_n$
- Iterate to find least n such that $\exists s \in \mathcal{S}_n.\ \neg(p(s))$
- Use SAT to find b_n such that $b_n \in \mathcal{S}_n \wedge \neg(p(b_n))$
- Use SAT to find b_{n-1} such that $b_{n-1} \in \mathcal{S}_{n-1} \wedge R\ b_{n-1}\ b_n$
- Use SAT to find b_{n-2} such that $b_{n-2} \in \mathcal{S}_{n-2} \wedge R\ b_{n-2}\ b_{n-1}$

$\vdots$

- Iterate to find b_0, b_1, ..., b_{n-1}, b_n where $b_i \in \mathcal{S}_i \wedge R\ b_{i-1}\ b_i$
- Then $b_0\ b_1\ \cdots\ b_{n-1}\ b_n$ is a path to a counterexample

Use SAT to find s_{n-1} such that $s_{n-1} \in \mathcal{S}_{n-1} \wedge R\ s_{n-1}\ s_n$

- Suppose states s, s' symbolically represented by $\vec{v}$, $\vec{v'}$
- Suppose BDD $\mathcal{B}_i$ represents $\vec{v} \in \mathcal{S}_i$ ($1 \leq i \leq n$)
- Suppose BDD $\mathcal{B}_R$ represents $R\ \vec{v}\ \vec{v'}$
- Then BDD
  $(\mathcal{B}_{n-1}\ \underline{\wedge}\ \mathcal{B}_R\underline{[\vec{b}_n/\vec{v'}]})$
  represents
  $\vec{v} \in \mathcal{S}_{n-1} \wedge R\ \vec{v}\ \vec{b}_n$
- Use SAT to find a valuation $\vec{b}_{n-1}$ for $\vec{v}$
- Then BDD
  $(\mathcal{B}_{n-1}\ \underline{\wedge}\ \mathcal{B}_R\underline{[\vec{b}_n/\vec{v'}]})\underline{[\vec{b}_{n-1}/\vec{v}]}$
  represents
  $\vec{b}_{n-1} \in \mathcal{S}_{n-1} \wedge R\ \vec{b}_{n-1}\ \vec{b}_n$

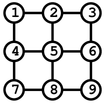
Generating counterexamples with BDDs

BDD algorithms can find satisfying assignments (SAT)

- $M = (S, S_0, R, L)$ and $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_M, \mathcal{B}_R, \mathcal{B}_p$ as earlier
- Suppose $\mathcal{B}_M \underline{\Rightarrow} \mathcal{B}_p$ is not $\boxed{1}$
- Must exist a state $s \in$ Reachable M such that $\neg(p\ s)$
- Let $\mathcal{B}_{\neg p}$ be the BDD representing $\neg(p\ \vec{v})$
- Iterate to find first n such that $\mathcal{B}_n \underline{\wedge} \mathcal{B}_{\neg p}$
- Use SAT to find $\vec{b}_n$ such that $(\mathcal{B}_n \underline{\wedge} \mathcal{B}_{\neg p})\underline{[\vec{b}_n/\vec{v}]}$
- Use SAT to find $\vec{b}_{n-1}$ such that $(\mathcal{B}_{n-1} \underline{\wedge} \mathcal{B}_R\underline{[\vec{b}_n/\vec{v}']})\underline{[\vec{b}_{n-1}/\vec{v}]}$
- For $0 < i < n$ find $\vec{b}_{i-1}$ such that $(\mathcal{B}_{i-1} \underline{\wedge} \mathcal{B}_R\underline{[\vec{b}_i/\vec{v}']})\underline{[\vec{b}_{i-1}/\vec{v}]}$
- $\vec{b}_0, \ldots, \vec{b}_i, \ldots, \vec{b}_n$ is a counterexample trace
- Sometimes can use partitioning to avoid constructing $\mathcal{B}_R$

Example (from an exam)

Consider a 3x3 array of 9 switches



Suppose each switch 1,2,...,9 can either be on or off, and that toggling any switch will automatically toggle all its immediate neighbours. For example, toggling switch 5 will also toggle switches 2, 4, 6 and 8, and toggling switch 6 will also toggle switches 3, 5 and 9.

(a) Devise a state space [4 marks] and transition relation [6 marks] to represent the behaviour of the array of switches

You are given the problem of getting from an initial state in which even numbered switches are on and odd numbered switches are off, to a final state in which all the switches are off.

(b) Write down predicates on your state space that characterises the initial [2 marks] and final [2 marks] states.

(c) Explain how you might use a model checker to find a sequences of switches to toggle to get from the initial to final state. [6 marks]

You are not expected to actually solve the problem, but only to explain how to represent it in terms of model checking.

Solution

A state is a vector `(v1,v2,v3,v4,v5,v6,v7,v8,v9)`, where $\texttt{vi} \in \mathbb{B}$

A transition relation `Trans` is then defined by:

```
Trans(v1,v2,v3,v4,v5,v6,v7,v8,v9)(v1′,v2′,v3′,v4′,v5′,v6′,v7′,v8′,v9′)
 =  ((v1′=¬v1)∧(v2′=¬v2)∧(v3′=v3)∧(v4′=¬v4)∧(v5′=v5)∧
     (v6′=v6)∧(v7′=v7)∧(v8′=v8)∧(v9′=v9))        (toggle switch 1)
 ∨  ((v1′=¬v1)∧(v2′=¬v2)∧(v3′=¬v3)∧(v4′=v4)∧(v5′=¬v5)∧
      (v6′=v6)∧(v7′=v7)∧(v8′=v8)∧(v9′=v9))       (toggle switch 2)
 ∨  ((v1′=v1)∧(v2′=¬v2)∧(v3′=¬v3)∧(v4′=v4)∧(v5′=v5)∧
      (v6′=¬v6)∧(v7′=v7)∧(v8′=v8)∧(v9′=v9))      (toggle switch 3)
 ∨  ((v1′=¬v1)∧(v2′=v2)∧(v3′=v3)∧(v4′=¬v4)∧(v5′=¬v5)∧
      (v6′=v6)∧(v7′=¬v7)∧(v8′=v8)∧(v9′=v9))      (toggle switch 4)
 ∨  ((v1′=v1)∧(v2′=¬v2)∧(v3′=v3)∧(v4′=¬v4)∧(v5′=¬v5)∧
      (v6′=¬v6)∧(v7′=v7)∧(v8′=¬v8)∧(v9′=v9))     (toggle switch 5)
 ∨  ((v1′=v1)∧(v2′=v2)∧(v3′=¬v3)∧(v4′=v4)∧(v5′=¬v5)∧
      (v6′=¬v6)∧(v7′=v7)∧(v8′=v8)∧(v9′=¬v9))     (toggle switch 6)
 ∨  ((v1′=v1)∧(v2′=v2)∧(v3′=v3)∧(v4′=¬v4)∧(v5′=v5)∧
      (v6′=v6)∧(v7′=¬v7)∧(v8′=¬v8)∧(v9′=v9))     (toggle switch 7)
 ∨  ((v1′=v1)∧(v2′=v2)∧(v3′=v3)∧(v4′=v4)∧(v5′=¬v5)∧
      (v6′=v6)∧(v7′=¬v7)∧(v8′=¬v8)∧(v9′=¬v9))    (toggle switch 8)
 ∨  ((v1′=v1)∧(v2′=v2)∧(v3′=v3)∧(v4′=v4)∧(v5′=v5)∧
      (v6′=¬v6)∧(v7′=v7)∧(v8′=¬v8)∧(v9′=¬v9))    (toggle switch 9)
```

Solution (continued)

Predicates `Init`, `Final` characterising the initial and final states, respectively, are defined by:

```
Init(v1,v2,v3,v4,v5,v6,v7,v8,v9) =
 ¬v1 ∧ v2 ∧ ¬v3 ∧ v4 ∧ ¬v5 ∧ v6 ∧ ¬v7 ∧ v8 ∧ ¬v9

Final(v1,v2,v3,v4,v5,v6,v7,v8,v9) =
 ¬v1 ∧ ¬v2 ∧ ¬v3 ∧ ¬v4 ∧ ¬v5 ∧ ¬v6 ∧ ¬v7 ∧ ¬v8 ∧ ¬v9
```

Model checkers can find counter-examples to properties, and sequences of transitions from an initial state to a counter-example state. Thus we could use a model checker to find a trace to a counter-example to the property that

```
¬Final(v1,v2,v3,v4,v5,v6,v7,v8,v9)
```

More Interesting Properties (1): LTL

More General Properties

- $\forall s{\in}S_0.\forall s'.R^*\ s\ s' \Rightarrow p\ s'$ says p true in all reachable states
- Might want to verify other properties
  1. `DeviceEnabled` holds infinitely often along every path
  2. From any state it is possible to get to a state where `Restart` holds
  3. After a three or more consecutive occurrences of `Req` there will eventually be an `Ack`
- Temporal logic can express such properties
- There are several temporal logics in use
  - LTL is good for the first example above
  - CTL is good for the second example
  - PSL is good for the third example
- Model checking:
  - Emerson, Clarke & Sifakis: Turing Award 2008
  - widely used in industry: first hardware, later software

Temporal logic selected history

Prior (1914-1969) devised ‘tense logic’ for investigating: “the relationship between tense and modality attributed to the Megarian philosopher Diodorus Cronus (ca. 340-280 BCE)”.

More details:
http://plato.stanford.edu/entries/prior/

- Temporal logic: deductive system for reasoning about time
  - temporal formulae for expressing temporal statements
  - deductive system for proving theorems
- Temporal logic model checking
  - uses semantics to check truth of temporal formulae in models
- Temporal logic *proof systems* are also of interest (but not in this course).

Temporal logic selected history (2)

- Many different languages capturing temporal statements as formulae
  - linear time (LTL)
  - branching time (CTL)
  - finite intervals (SEREs)
  - industrial languages (PSL, SVA)
- Prior used linear time, Kripke suggested branching time:

  ... we perhaps should not regard time as a linear series ... there are several possibilities for what the next moment may be like - and for each possible next moment, there are several possibilities for the moment after that. Thus the situation takes the form, not of a linear sequence, but of a 'tree'. [Saul Kripke, 1958 (aged 17, still at school)]
- CS issues different from philosophical issues
  - Moshe Vardi: “Branching vs. Linear Time: Final Showdown”

http://www.computer.org/portal/web/awards/Vardi

Moshe Vardi
www.computer.org

"For fundamental and lasting contributions to the development of logic as a unifying foundational framework and a tool for modeling computational systems"

2011 Harry H. Goode Memorial Award Recipient

Linear Temporal Logic (LTL)

- Grammar of *LTL formulae* ϕ

$$\begin{array}{lcll} \phi & ::= & p & \text{(Atomic formula: } p \in AP\text{)} \\ & | & \neg\phi & \text{(Negation)} \\ & | & \phi_1 \vee \phi_2 & \text{(Disjunction)} \\ & | & \mathbf{X}\phi & \text{(successor)} \\ & | & \mathbf{F}\phi & \text{(sometimes)} \\ & | & \mathbf{G}\phi & \text{(always)} \\ & | & [\phi_1\ \mathbf{U}\ \phi_2] & \text{(Until)} \end{array}$$

- **G**: (G)lobally holds; **F**: At some point in the (F)uture; **X**: In the ne(X)t state
- Details differ from Prior's tense logic – but similar ideas
- Semantics define when ϕ true in model M
  - where $M = (S, S_0, R, L)$ – a Kripke structure
  - notation: $M \models \phi$ means ϕ true in model M
  - model checking algorithms compute this (when decidable)
  - previously we only discussed the case $\phi = \mathbf{AG}\, p$

While use temporal operators at all?

Instead of the complexity of new temporal operators, why not make time explicit and just write:

- $\exists t.\phi(t)$ instead of $\mathbf{F}\phi$
- $\forall t.\phi(t)$ instead of $\mathbf{G}\phi$
- $\phi[t+1/t]$ instead of $\mathbf{X}\phi$

along with parameterising all Atomic Formulae with time?

Answer: it's harder to reason about quantifiers and arithmetic on time than it is to reason about temporal operators (which abstract from the above concrete notion of time).

$M \models \phi$ means "formula ϕ is true in model M"

- If $M = (S, S_0, R, L)$ then

  $\boxed{\pi \text{ is an } M\text{-path starting from } s \text{ iff } \mathrm{Path}\ R\ s\ \pi}$

- If $M = (S, S_0, R, L)$ then we define $M \models \phi$ to mean:

  $\boxed{\phi \text{ is true on all } M\text{-paths starting from a member of } S_0}$

- We will define $[\![\phi]\!]_M(\pi)$ to mean

  $\boxed{\phi \text{ is true on the } M\text{-path } \pi}$

- Thus $M \models \phi$ will be formally defined by:

  $\boxed{M \models \phi \ \Leftrightarrow\ \forall \pi\ s.\ s \in S_0 \wedge \mathrm{Path}\ R\ s\ \pi \Rightarrow [\![\phi]\!]_M(\pi)}$

- It remains to actually define $[\![\phi]\!]_M$ for all formulae ϕ

Definition of $[\![\phi]\!]_M(\pi)$

- $[\![\phi]\!]_M(\pi)$ is the application of function $[\![\phi]\!]_M$ to path π
  - thus $[\![\phi]\!]_M : (\mathbb{N} \to S) \to \mathbb{B}$
- Let $M = (S, S_0, R, L)$

  $[\![\phi]\!]_M$ is defined by structural induction on ϕ

$$
\begin{array}{lcl}
[\![p]\!]_M(\pi) & = & p \in L(\pi\ 0) \\
[\![\neg\phi]\!]_M(\pi) & = & \neg([\![\phi]\!]_M(\pi)) \\
[\![\phi_1 \vee \phi_2]\!]_M(\pi) & = & [\![\phi_1]\!]_M(\pi)\ \vee\ [\![\phi_2]\!]_M(\pi) \\
[\![\mathbf{X}\phi]\!]_M(\pi) & = & [\![\phi]\!]_M(\pi{\downarrow}1) \\
[\![\mathbf{F}\phi]\!]_M(\pi) & = & \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \\
[\![\mathbf{G}\phi]\!]_M(\pi) & = & \forall i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \\
[\![[\phi_1\ \mathbf{U}\ \phi_2]]\!]_M(\pi) & = & \exists i.\ [\![\phi_2]\!]_M(\pi{\downarrow}i)\ \wedge\ \forall j.\ j{<}i\ \Rightarrow\ [\![\phi_1]\!]_M(\pi{\downarrow}j)
\end{array}
$$

- We look at each of these semantic equations in turn

$[\![p]\!]_M(\pi) = p(\pi\ 0)$

- Assume $M = (S, S_0, R, L)$
- We have: $[\![p]\!]_M(\pi) = p \in L(\pi\ 0)$
  - p is an atomic property, i.e. $p \in AP$
  - $\pi : \mathbb{N} \to S$ so $\pi\ 0 \in S$
  - $\pi\ 0$ is the first state in path π
  - $p \in L(\pi\ 0)$ is *true* iff atomic property p holds of state $\pi\ 0$
- $[\![p]\!]_M(\pi)$ means p holds of the first state in path π
- $\mathtt{T}, \mathtt{F} \in AP$ with $\mathtt{T} \in L(s)$ and $\mathtt{F} \notin L(s)$ for all $s \in S$
  - $[\![\mathtt{T}]\!]_M(\pi)$ is always true
  - $[\![\mathtt{F}]\!]_M(\pi)$ is always false

$$[\![\neg\phi]\!]_M(\pi) = \neg([\![\phi]\!]_M(\pi))$$
$$[\![\phi_1 \vee \phi_2]\!]_M(\pi) = [\![\phi_1]\!]_M(\pi) \vee [\![\phi_2]\!]_M(\pi)$$

- $[\![\neg\phi]\!]_M(\pi) = \neg([\![\phi]\!]_M(\pi))$
  - $[\![\neg\phi]\!]_M(\pi)$ true iff $[\![\phi]\!]_M(\pi)$ is not true

- $[\![\phi_1 \vee \phi_2]\!]_M(\pi) = [\![\phi_1]\!]_M(\pi) \vee [\![\phi_2]\!]_M(\pi)$
  - $[\![\phi_1 \vee \phi_2]\!]_M(\pi)$ true iff $[\![\phi_1]\!]_M(\pi)$ is true or $[\![\phi_2]\!]_M(\pi)$ is true

$[\![\mathbf{X}\phi]\!]_M(\pi) = [\![\phi]\!]_M(\pi{\downarrow}1)$

- $[\![\mathbf{X}\phi]\!]_M(\pi) = [\![\phi]\!]_M(\pi{\downarrow}1)$
  - $\pi{\downarrow}1$ is π with the first state chopped off

$$\begin{aligned}\pi{\downarrow}1(0) &= \pi(1+0) = \pi(1)\\ \pi{\downarrow}1(1) &= \pi(1+1) = \pi(2)\\ \pi{\downarrow}1(2) &= \pi(1+2) = \pi(3)\\ &\vdots\end{aligned}$$

- $[\![\mathbf{X}\phi]\!]_M(\pi)$ true iff $[\![\phi]\!]_M$ true *starting at the second state of* π

$[\![\mathbf{F}\phi]\!]_M(\pi) = \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i)$

- $[\![\mathbf{F}\phi]\!]_M(\pi) = \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i)$
  - $\pi{\downarrow}i$ is π with the first i states chopped off

    $$\begin{aligned} \pi{\downarrow}i(0) &= \pi(i+0) = \pi(i) \\ \pi{\downarrow}i(1) &= \pi(i+1) \\ \pi{\downarrow}i(2) &= \pi(i+2) \\ &\ \vdots \end{aligned}$$
  - $[\![\phi]\!]_M(\pi{\downarrow}i)$ true iff $[\![\phi]\!]_M$ true *starting i states along* π

- $[\![\mathbf{F}\phi]\!]_M(\pi)$ true iff $[\![\phi]\!]_M$ true *starting somewhere along* π

- "$\mathbf{F}\phi$" is read as "sometimes ϕ"

$[\![\mathbf{G}\phi]\!]_M(\pi) = \forall i.\ [\![\phi]\!]_M(\pi{\downarrow}i)$

- $[\![\mathbf{G}\phi]\!]_M(\pi) = \forall i.\ [\![\phi]\!]_M(\pi{\downarrow}i)$
  - $\pi{\downarrow}i$ is π with the first i states chopped off
  - $[\![\phi]\!]_M(\pi{\downarrow}i)$ true iff $[\![\phi]\!]_M$ true *starting i states along* π
- $[\![\mathbf{G}\phi]\!]_M(\pi)$ true iff $[\![\phi]\!]_M$ true *starting anywhere along* π
- "$\mathbf{G}\phi$" is read as "always ϕ" or "globally ϕ"
- $M \models \mathbf{AG}\,p$ defined earlier: $M \models \mathbf{AG}\,p \Leftrightarrow M \models \mathbf{G}(p)$
- $\mathbf{G}$ is definable in terms of $\mathbf{F}$ and $\neg$: $\mathbf{G}\phi = \neg(\mathbf{F}(\neg\phi))$

$$\begin{aligned}
[\![\neg(\mathbf{F}(\neg\phi))]\!]_M(\pi) &= \neg([\![\mathbf{F}(\neg\phi)]\!]_M(\pi)) \\
&= \neg(\exists i.\ [\![\neg\phi]\!]_M(\pi{\downarrow}i)) \\
&= \neg(\exists i.\ \neg([\![\phi]\!]_M(\pi{\downarrow}i))) \\
&= \forall i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \\
&= [\![\mathbf{G}\phi]\!]_M(\pi)
\end{aligned}$$

$\boxed{[\![[\phi_1 \ \mathbf{U}\ \phi_2]]\!]_M(\pi) = \exists i.\ [\![\phi_2]\!]_M(\pi{\downarrow}i) \wedge \forall j.\ j{<}i \Rightarrow [\![\phi_1]\!]_M(\pi{\downarrow}j)}$

- $[\![[\phi_1 \ \mathbf{U}\ \phi_2]]\!]_M(\pi) = \exists i.\ [\![\phi_2]\!]_M(\pi{\downarrow}i) \wedge \forall j.\ j{<}i \Rightarrow [\![\phi_1]\!]_M(\pi{\downarrow}j)$
  - $[\![\phi_2]\!]_M(\pi{\downarrow}i)$ true iff $[\![\phi_2]\!]_M$ true *starting i states along π*
  - $[\![\phi_1]\!]_M(\pi{\downarrow}j)$ true iff $[\![\phi_1]\!]_M$ true *starting j states along π*

- $[\![[\phi_1 \ \mathbf{U}\ \phi_2]]\!]_M(\pi)$ is true iff

  $[\![\phi_2]\!]_M$ is true somewhere along π and up to then $[\![\phi_1]\!]_M$ is true

- "$[\phi_1 \ \mathbf{U}\ \phi_2]$" is read as "$\phi_1$ until ϕ_2"

- **F** is definable in terms of $[- \ \mathbf{U}\ -]$: $\mathbf{F}\phi = [\top \ \mathbf{U}\ \phi]$

$$
\begin{aligned}
& [\![[\top \ \mathbf{U}\ \phi]]\!]_M(\pi) \\
&= \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \wedge \forall j.\ j{<}i \Rightarrow [\![\top]\!]_M(\pi{\downarrow}j) \\
&= \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \wedge \forall j.\ j{<}i \Rightarrow \mathit{true} \\
&= \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \wedge \mathit{true} \\
&= \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \\
&= [\![\mathbf{F}\phi]\!]_M(\pi)
\end{aligned}
$$

Review of Linear Temporal Logic (LTL)

- Grammar of *LTL formulae* ϕ (slide 64)

ϕ	$::=$	p	(Atomic formula: $p \in AP$)
	$\mid$	$\neg\phi$	(Negation)
	$\mid$	$\phi_1 \vee \phi_2$	(Disjunction)
	$\mid$	$\mathbf{X}\phi$	(successor)
	$\mid$	$\mathbf{F}\phi$	(sometimes)
	$\mid$	$\mathbf{G}\phi$	(always)
	$\mid$	$[\phi_1\ \mathbf{U}\ \phi_2]$	(Until)

- $M \models \phi$ means ϕ holds on all M-paths
  - $M = (S, S_0, R, L)$
  - $[\![\phi]\!]_M(\pi)$ means ϕ is true on the M-path π
  - $M \models \phi \ \Leftrightarrow\ \forall\pi\ s.\ s \in S_0 \wedge \mathrm{Path}\ R\ s\ \pi \Rightarrow [\![\phi]\!]_M(\pi)$

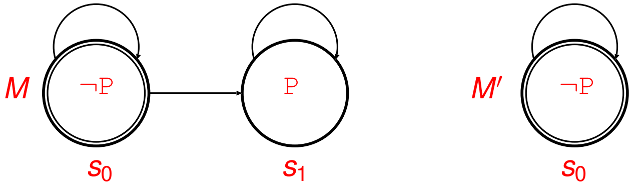
LTL examples

- “`DeviceEnabled` holds infinitely often along every path”
  $\boxed{\mathbf{G}(\mathbf{F}\ \texttt{DeviceEnabled})}$
- “Eventually the state becomes permanently `Done`“
  $\boxed{\mathbf{F}(\mathbf{G}\ \texttt{Done})}$
- “Every `Req` is followed by an `Ack`”
  $\boxed{\mathbf{G}(\texttt{Req} \Rightarrow \mathbf{F}\ \texttt{Ack})}$
  Number of `Req` and `Ack` may differ - no counting
- “If `Enabled` infinitely often then `Running` infinitely often”
  $\boxed{\mathbf{G}(\mathbf{F}\ \texttt{Enabled}) \Rightarrow \mathbf{G}(\mathbf{F}\ \texttt{Running})}$
- “An upward-going lift at the second floor keeps going up if a passenger requests the fifth floor”
  $$\boxed{\begin{array}{l}\mathbf{G}(\texttt{AtFloor2} \wedge \texttt{DirectionUp} \wedge \texttt{RequestFloor5} \\ \quad \Rightarrow [\texttt{DirectionUp}\ \mathbf{U}\ \texttt{AtFloor5}])\end{array}}$$

(acknowledgement: `http://pswlab.kaist.ac.kr/courses/cs402-2011/temporal-logic2.pdf`)

A property not expressible in LTL

- Let $AP = \{\mathtt{P}\}$ and consider models M and M' below



$$\begin{array}{lcl} M & = & (\{s_0, s_1\}, \{s_0\}, \{(s_0, s_0), (s_0, s_1), (s_1, s_1)\}, L) \\ M' & = & (\{s_0\}, \{s_0\}, \{(s_0, s_0)\}, L) \end{array}$$

where: $L \ = \ \lambda s.\ \mathit{if}\ s = s_0\ \mathit{then}\ \{\}\ \mathit{else}\ \{\mathtt{P}\}$

- Every M'-path is also an M-path
- So if ϕ true on every M-path then ϕ true on every M'-path
- Hence in LTL for any ϕ if $M \models \phi$ then $M' \models \phi$
- Consider $\phi_{\mathtt{P}} \Leftrightarrow$ "can always reach a state satisfying $\mathtt{P}$"
  - $\phi_{\mathtt{P}}$ holds in M but not in M'
  - but in LTL can't have $M \models \phi_{\mathtt{P}}$ and not $M' \models \phi_{\mathtt{P}}$
- hence $\phi_{\mathtt{P}}$ not expressible in LTL

(acknowledgement: *Logic in Computer Science*, Huth & Ryan (2nd Ed.) page 219, ISBN 0 521 54310 X)

LTL expressibility limitations

“can always reach a state satisfying `p`”

- In LTL $M \models \phi$ says ϕ holds of all paths of M
- LTL formulae ϕ are *evaluated on paths* .... path formulae
- Want also to say that from any state there exists a path to some state satisfying p
  - $\forall s.\ \exists \pi.\ \mathrm{Path}\ R\ s\ \pi\ \wedge\ \exists i.\ p \in L(\pi(i))$
  - but this isn’t expressible in LTL (see slide 76)

By contrast:

- CTL properties are *evaluated at a state* ... state formulae
  - they can talk about both some or all paths
  - starting from the state they are evaluated at

More Interesting Properties (2): CTL

Computation Tree Logic (CTL)

- LTL formulae ϕ are evaluated on paths .... path formulae
- CTL formulae ψ are evaluated on states .. state formulae

- Syntax of CTL well-formed formulae:

$$
\begin{array}{rcll}
\psi & ::= & p & \text{(Atomic formula } p \in AP\text{)} \\
 & | & \neg\psi & \text{(Negation)} \\
 & | & \psi_1 \wedge \psi_2 & \text{(Conjunction)} \\
 & | & \psi_1 \vee \psi_2 & \text{(Disjunction)} \\
 & | & \psi_1 \Rightarrow \psi_2 & \text{(Implication)} \\
 & | & \mathbf{AX}\psi & \text{(All successors)} \\
 & | & \mathbf{EX}\psi & \text{(Some successors)} \\
 & | & \mathbf{A}[\psi_1 \ \mathbf{U} \ \psi_2] & \text{(Until -- along all paths)} \\
 & | & \mathbf{E}[\psi_1 \ \mathbf{U} \ \psi_2] & \text{(Until -- along some path)}
\end{array}
$$

- (Some operators can be defined in terms of others)

Semantics of CTL

- Assume $M = (S, S_0, R, L)$ and then define:

$$
\begin{array}{lcl}
[\![p]\!]_M(s) & = & p \in L(s) \\
[\![\neg\psi]\!]_M(s) & = & \neg([\![\psi]\!]_M(s)) \\
[\![\psi_1 \wedge \psi_2]\!]_M(s) & = & [\![\psi_1]\!]_M(s) \ \wedge \ [\![\psi_2]\!]_M(s) \\
[\![\psi_1 \vee \psi_2]\!]_M(s) & = & [\![\psi_1]\!]_M(s) \ \vee \ [\![\psi_2]\!]_M(s) \\
[\![\psi_1 \Rightarrow \psi_2]\!]_M(s) & = & [\![\psi_1]\!]_M(s) \ \Rightarrow \ [\![\psi_2]\!]_M(s) \\
[\![\mathbf{AX}\psi]\!]_M(s) & = & \forall s'.\ R\ s\ s' \ \Rightarrow \ [\![\psi]\!]_M(s') \\
[\![\mathbf{EX}\psi]\!]_M(s) & = & \exists s'.\ R\ s\ s' \ \wedge \ [\![\psi]\!]_M(s') \\
[\![\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_M(s) & = & \forall \pi.\ \mathrm{Path}\ R\ s\ \pi \\
 & & \quad \Rightarrow \exists i.\ [\![\psi_2]\!]_M(\pi(i)) \\
 & & \qquad\qquad \wedge \\
 & & \qquad\qquad \forall j.\ j{<}i \ \Rightarrow \ [\![\psi_1]\!]_M(\pi(j)) \\
[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_M(s) & = & \exists \pi.\ \mathrm{Path}\ R\ s\ \pi \\
 & & \quad \wedge \ \exists i.\ [\![\psi_2]\!]_M(\pi(i)) \\
 & & \qquad\qquad \wedge \\
 & & \qquad\qquad \forall j.\ j{<}i \ \Rightarrow \ [\![\psi_1]\!]_M(\pi(j))
\end{array}
$$

The defined operator **AF**

- Define $\mathbf{AF}\psi = \mathbf{A}[\mathtt{T}\ \mathbf{U}\ \psi]$

- $\mathbf{AF}\psi$ true at s iff ψ true somewhere on every R-path from s

$$
\begin{aligned}
[\![\mathbf{AF}\psi]\!]_M(s) &= [\![\mathbf{A}[\mathtt{T}\ \mathbf{U}\ \psi]]\!]_M(s) \\
&= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \\
&\qquad \Rightarrow \\
&\qquad \exists i.\ [\![\psi]\!]_M(\pi(i)) \ \wedge\ \forall j.\ j < i \ \Rightarrow\ [\![\mathtt{T}]\!]_M(\pi(j)) \\
&= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \\
&\qquad \Rightarrow \\
&\qquad \exists i.\ [\![\psi]\!]_M(\pi(i)) \ \wedge\ \forall j.\ j < i \ \Rightarrow\ \mathit{true} \\
&= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \ \Rightarrow\ \exists i.\ [\![\psi]\!]_M(\pi(i))
\end{aligned}
$$

The defined operator **EF**

- Define $\textbf{EF}\psi = \textbf{E}[\top\ \textbf{U}\ \psi]$
- $\textbf{EF}\psi$ true at s iff ψ true somewhere on some R-path from s

$$
\begin{aligned}
[\![\textbf{EF}\psi]\!]_M(s) &= [\![\textbf{E}[\top\ \textbf{U}\ \psi]]\!]_M(s) \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \\
&\qquad \wedge \\
&\qquad \exists i.\ [\![\psi]\!]_M(\pi(i)) \ \wedge\ \forall j.\ j < i \ \Rightarrow\ [\![\top]\!]_M(\pi(j)) \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \\
&\qquad \wedge \\
&\qquad \exists i.\ [\![\psi]\!]_M(\pi(i)) \ \wedge\ \forall j.\ j < i \ \Rightarrow\ \textit{true} \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \ \wedge\ \exists i.\ [\![\psi]\!]_M(\pi(i))
\end{aligned}
$$

- "can reach a state satisfying p" is $\textbf{EF}\ p$

The defined operator **AG**

- Define $\mathbf{AG}\psi = \neg\mathbf{EF}(\neg\psi)$
- $\mathbf{AG}\psi$ true at s iff ψ true everywhere on every R-path from s

$$\begin{aligned}
[\![\mathbf{AG}\psi]\!]_M(s) &= [\![\neg\mathbf{EF}(\neg\psi)]\!]_M(s) \\
&= \neg([\![\mathbf{EF}(\neg\psi)]\!]_M(s)) \\
&= \neg(\exists\pi.\ \mathrm{Path}\ R\ s\ \pi \wedge \exists i.\ [\![\neg\psi]\!]_M(\pi(i))) \\
&= \neg(\exists\pi.\ \mathrm{Path}\ R\ s\ \pi \wedge \exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \forall\pi.\ \neg(\mathrm{Path}\ R\ s\ \pi \wedge \exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \forall\pi.\ \neg\mathrm{Path}\ R\ s\ \pi \vee \neg(\exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \forall\pi.\ \neg\mathrm{Path}\ R\ s\ \pi \vee \forall i.\ \neg\neg[\![\psi]\!]_M(\pi(i)) \\
&= \forall\pi.\ \neg\mathrm{Path}\ R\ s\ \pi \vee \forall i.\ [\![\psi]\!]_M(\pi(i)) \\
&= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \Rightarrow \forall i.\ [\![\psi]\!]_M(\pi(i))
\end{aligned}$$

- $\mathbf{AG}\psi$ means ψ true at all reachable states
- $[\![\mathbf{AG}(p)]\!]_M(s) \ \equiv\ \forall s'.\ R^*\ s\ s' \ \Rightarrow\ p \in L(s')$
- "can always reach a state satisfying p" is $\mathbf{AG}(\mathbf{EF}\ p)$

The defined operator **EG**

- Define $\mathbf{EG}\psi = \neg\mathbf{AF}(\neg\psi)$

- $\mathbf{EG}\psi$ true at s iff ψ true everywhere on some R-path from s

$$\begin{aligned}
[\![\mathbf{EG}\psi]\!]_M(s) &= [\![\neg\mathbf{AF}(\neg\psi)]\!]_M(s) \\
&= \neg([\![\mathbf{AF}(\neg\psi)]\!]_M(s)) \\
&= \neg(\forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow \exists i.\ [\![\neg\psi]\!]_M(\pi(i))) \\
&= \neg(\forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow \exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \exists\pi.\ \neg(\text{Path}\ R\ s\ \pi \Rightarrow \exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \wedge \neg(\exists i.\ \neg[\![\psi]\!]_M(\pi(i))) \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \wedge \forall i.\ \neg\neg[\![\psi]\!]_M(\pi(i)) \\
&= \exists\pi.\ \text{Path}\ R\ s\ \pi \wedge \forall i.\ [\![\psi]\!]_M(\pi(i))
\end{aligned}$$

The defined operator $\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]$

- $\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]$ is a 'partial correctness' version of $\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]$
- It is true at s if along all R-paths from s:
  - ψ_1 always holds on the path, or
  - ψ_2 holds sometime on the path, and until it does ψ_1 holds
- Define

$$
\begin{array}{l}
[\![\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]]\!]_M(s) \\
\quad = [\![\neg\mathbf{E}[(\psi_1\wedge\neg\psi_2)\ \mathbf{U}\ (\neg\psi_1\wedge\neg\psi_2)]]\!]_M(s) \\
\quad = \neg[\![\mathbf{E}[(\psi_1\wedge\neg\psi_2)\ \mathbf{U}\ (\neg\psi_1\wedge\neg\psi_2)]]\!]_M(s) \\
\quad = \neg(\exists\pi.\ \mathrm{Path}\ R\ s\ \pi \\
\qquad\qquad\quad \wedge \\
\qquad\qquad\quad \exists i.\ [\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i)) \\
\qquad\qquad\qquad \wedge \\
\qquad\qquad\qquad \forall j.\ j<i\ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j)))
\end{array}
$$

- Exercise: understand the next two slides!

$\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]$ continued (1)

- Continuing:

$$
\begin{array}{ll}
& \neg(\exists\pi.\ \mathsf{Path}\ R\ s\ \pi \\
& \qquad \wedge \\
& \qquad \exists i.\ [\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i))\ \wedge\ \forall j.\ j<i\ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j))) \\
= & \forall\pi.\ \neg(\mathsf{Path}\ R\ s\ \pi \\
& \qquad \wedge \\
& \qquad \exists i.\ [\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i))\ \wedge\ \forall j.\ j<i\ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j))) \\
= & \forall\pi.\ \mathsf{Path}\ R\ s\ \pi \\
& \qquad \Rightarrow \\
& \qquad \neg(\exists i.\ [\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i))\ \wedge\ \forall j.\ j<i\ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j))) \\
= & \forall\pi.\ \mathsf{Path}\ R\ s\ \pi \\
& \qquad \Rightarrow \\
& \qquad \forall i.\ \neg[\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i)) \vee \neg(\forall j.\ j<i\ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j)))
\end{array}
$$

$\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]$ continued (2)

- Continuing:

$$= \forall\pi.\ \mathsf{Path}\ R\ s\ \pi \Rightarrow \forall i.\ \neg[\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i)) \vee \neg(\forall j.\ j<i \ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j)))$$

$$= \forall\pi.\ \mathsf{Path}\ R\ s\ \pi \Rightarrow \forall i.\ \neg(\forall j.\ j<i \ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j))) \vee \neg[\![\neg\psi_1\wedge\neg\psi_2]\!]_M(\pi(i))$$

$$= \forall\pi.\ \mathsf{Path}\ R\ s\ \pi \Rightarrow \forall i.\ (\forall j.\ j<i \ \Rightarrow\ [\![\psi_1\wedge\neg\psi_2]\!]_M(\pi(j))) \ \Rightarrow\ [\![\psi_1\vee\psi_2]\!]_M(\pi(i))$$

- Exercise: explain why this is $[\![\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]]\!]_M(s)$?
  - this exercise illustrates the subtlety of writing CTL!

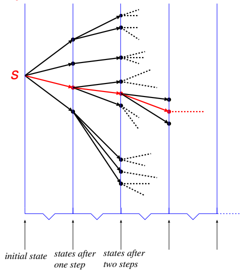
Sanity check: $\mathbf{A}[\psi\ \mathbf{W}\ \mathtt{F}] = \mathbf{AG}\ \psi$

- From last slide:
  $[\![\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]]\!]_M(s)$
  $= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi$
  $\Rightarrow \forall i.\ (\forall j.\ j{<}i \Rightarrow [\![\psi_1{\wedge}\neg\psi_2]\!]_M(\pi(j))) \Rightarrow [\![\psi_1{\vee}\psi_2]\!]_M(\pi(i))$
- Set ψ_1 to ψ and ψ_2 to $\mathtt{F}$:
  $[\![\mathbf{A}[\psi\ \mathbf{W}\ \mathtt{F}]]\!]_M(s)$
  $= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi$
  $\Rightarrow \forall i.\ (\forall j.\ j{<}i \ \Rightarrow\ [\![\psi{\wedge}\neg\mathtt{F}]\!]_M(\pi(j)))\ \Rightarrow\ [\![\psi{\vee}\mathtt{F}]\!]_M(\pi(i))$
- Simplify:
  $[\![\mathbf{A}[\psi\ \mathbf{W}\ \mathtt{F}]]\!]_M(s)$
  $= \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \Rightarrow \forall i.\ (\forall j.\ j{<}i \Rightarrow [\![\psi]\!]_M(\pi(j))) \Rightarrow [\![\psi]\!]_M(\pi(i))$
- By induction on i:
  $[\![\mathbf{A}[\psi\ \mathbf{W}\ \mathtt{F}]]\!]_M(s)\ =\ \forall\pi.\ \mathrm{Path}\ R\ s\ \pi\ \Rightarrow\ \forall i.\ [\![\psi]\!]_M(\pi(i))$

- Exercises
  1. Describe the property: $\mathbf{A}[\mathtt{T}\ \mathbf{W}\ \psi]$.
  2. Describe the property: $\neg\mathbf{E}[\neg\psi_2\ \mathbf{U}\ \neg(\psi_1{\vee}\psi_2)]$.
  3. Define $\mathbf{E}[\psi_1\ \mathbf{W}\ \psi_2] = \mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2] \vee \mathbf{EG}\psi_1$.
     Describe the property: $\mathbf{E}[\psi_1\ \mathbf{W}\ \psi_2]$?

Recall model behaviour computation tree

- Atomic properties are true or false of individual states
- General properties are true or false of whole behaviour
- Behaviour of (S, R) starting from $s \in S$ as a tree:



- A path is shown in red
- Properties may look at all paths, or just a single path
  - CTL: Computation Tree Logic (all paths from a state)
  - LTL: Linear Temporal Logic (a single path)

Summary of CTL operators (primitive + defined)

- CTL formulae:

p	(Atomic formula - $p \in AP$)
$\neg\psi$	(Negation)
$\psi_1 \wedge \psi_2$	(Conjunction)
$\psi_1 \vee \psi_2$	(Disjunction)
$\psi_1 \Rightarrow \psi_2$	(Implication)
$\mathbf{AX}\psi$	(All successors)
$\mathbf{EX}\psi$	(Some successors)
$\mathbf{AF}\psi$	(Somewhere – along all paths)
$\mathbf{EF}\psi$	(Somewhere – along some path)
$\mathbf{AG}\psi$	(Everywhere – along all paths)
$\mathbf{EG}\psi$	(Everywhere – along some path)
$\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]$	(Until – along all paths)
$\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]$	(Until – along some path)
$\mathbf{A}[\psi_1\ \mathbf{W}\ \psi_2]$	(Unless – along all paths)
$\mathbf{E}[\psi_1\ \mathbf{W}\ \psi_2]$	(Unless – along some path)

Example CTL formulae

- $\mathbf{EF}(\mathit{Started} \wedge \neg \mathit{Ready})$

  It is possible to get to a state where Started holds but Ready does not hold

- $\mathbf{AG}(\mathit{Req} \Rightarrow \mathbf{AF}\mathit{Ack})$

  If a request Req occurs, then it will eventually be acknowledged by Ack

- $\mathbf{AG}(\mathbf{AF}\mathit{DeviceEnabled})$

  DeviceEnabled is always true somewhere along every path starting anywhere: i.e. DeviceEnabled holds infinitely often along every path

- $\mathbf{AG}(\mathbf{EF}\mathit{Restart})$

  From any state it is possible to get to a state for which Restart holds

  Can't be expressed in LTL!

More CTL examples (1)

- $\mathbf{AG}(\mathit{Req} \Rightarrow \mathbf{A}[\mathit{Req}\ \mathbf{U}\ \mathit{Ack}])$

  If a request Req occurs, then it continues to hold, until it is eventually acknowledged

- $\mathbf{AG}(\mathit{Req} \Rightarrow \mathbf{AX}(\mathbf{A}[\neg\mathit{Req}\ \mathbf{U}\ \mathit{Ack}]))$

  Whenever Req is true either it must become false on the next cycle and remains false until Ack, or Ack must become true on the next cycle

  Exercise: is the **AX** necessary?

- $\mathbf{AG}(\mathit{Req} \Rightarrow (\neg\mathit{Ack} \Rightarrow \mathbf{AX}(\mathbf{A}[\mathit{Req}\ \mathbf{U}\ \mathit{Ack}])))$

  Whenever Req is true and Ack is false then Ack will eventually become true and until it does Req will remain true

  Exercise: is the **AX** necessary?

More CTL examples (2)

- $\mathbf{AG}(\mathit{Enabled} \Rightarrow \mathbf{AG}(\mathit{Start} \Rightarrow \mathbf{A}[\neg \mathit{Waiting}\ \mathbf{U}\ \mathit{Ack}]))$

  If Enabled is ever true then if Start is true in any subsequent state then Ack will eventually become true, and until it does Waiting will be false

- $\mathbf{AG}(\neg \mathit{Req}_1 \wedge \neg \mathit{Req}_2 \Rightarrow \mathbf{A}[\neg \mathit{Req}_1 \wedge \neg \mathit{Req}_2\ \mathbf{U}\ (\mathit{Start} \wedge \neg \mathit{Req}_2)])$

  Whenever Req_1 and Req_2 are false, they remain false until Start becomes true with Req_2 still false

- $\mathbf{AG}(\mathit{Req} \Rightarrow \mathbf{AX}(\mathit{Ack} \Rightarrow \mathbf{AF}\ \neg \mathit{Req}))$

  If Req is true and Ack becomes true one cycle later, then eventually Req will become false

Some abbreviations

- $\mathbf{AX}_i\ \psi \ \equiv\ \underbrace{\mathbf{AX}(\mathbf{AX}(\cdots(\mathbf{AX}\ \psi)\cdots))}_{i \text{ instances of } \mathbf{AX}}$

  ψ is true on all paths i units of time later

- $\mathbf{ABF}_{i..j}\ \psi \ \equiv\ \mathbf{AX}_i\,\underbrace{(\psi \vee \mathbf{AX}(\psi \vee\ \cdots\ \mathbf{AX}(\psi \vee \mathbf{AX}\ \psi)\cdots))}_{j-i \text{ instances of } \mathbf{AX}}$

  ψ is true on all paths sometime between i units of time later and j units of time later

- $\mathbf{AG}(\mathit{Req} \Rightarrow \mathbf{AX}(\mathit{Ack}_1 \wedge \mathbf{ABF}_{1..6}(\mathit{Ack}_2 \wedge \mathbf{A}[\mathit{Wait}\ \mathbf{U}\ \mathit{Reply}])))$

  One cycle after Req, Ack_1 should become true, and then Ack_2 becomes true 1 to 6 cycles later and then eventually Reply becomes true, but until it does Wait holds from the time of Ack_2

- More abbreviations in ‘Industry Standard’ language PSL

CTL model checking

- For LTL path formulae ϕ recall that $M \models \phi$ is defined by:

  $$\boxed{M \models \phi \;\Leftrightarrow\; \forall \pi\ s.\ s \in S_0 \wedge \mathrm{Path}\ R\ s\ \pi \Rightarrow [\![\phi]\!]_M(\pi)}$$

- For CTL state formulae ψ the definition of $M \models \psi$ is:

  $$\boxed{M \models \psi \;\Leftrightarrow\; \forall s.\ s \in S_0 \Rightarrow [\![\psi]\!]_M(s)}$$

- M common; LTL, CTL formulae and semantics $[\![\ ]\!]_M$ differ

- CTL model checking algorithm:
  - compute $\{s \mid [\![\psi]\!]_M(s) = \mathit{true}\}$ bottom up
  - check $S_0 \subseteq \{s \mid [\![\psi]\!]_M(s) = \mathit{true}\}$
  - symbolic model checking represents these sets as BDDs

CTL model checking: p, $\mathbf{AX}\psi$, $\mathbf{EX}\psi$

- For CTL formula ψ let $\{\!|\psi|\!\}_M = \{s \mid [\![\psi]\!]_M(s) = \mathit{true}\}$
- When unambiguous will write $\{\!|\psi|\!\}$ instead of $\{\!|\psi|\!\}_M$
- $\{\!|p|\!\} = \{s \mid p \in L(s)\}$
  - scan through set of states S marking states labelled with p
  - $\{\!|p|\!\}$ is set of marked states
- To compute $\{\!|\mathbf{AX}\psi|\!\}$
  - recursively compute $\{\!|\psi|\!\}$
  - marks those states all of whose successors are in $\{\!|\psi|\!\}$
  - $\{\!|\mathbf{AX}\psi|\!\}$ is the set of marked states
- To compute $\{\!|\mathbf{EX}\psi|\!\}$
  - recursively compute $\{\!|\psi|\!\}$
  - marks those states with at least one successor in $\{\!|\psi|\!\}$
  - $\{\!|\mathbf{EX}\psi|\!\}$ is the set of marked states

CTL model checking: $[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$, $[\![\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$

- To compute $[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$
  - recursively compute $[\![\psi_1]\!]$ and $[\![\psi_2]\!]$
  - mark all states in $[\![\psi_2]\!]$
  - mark all states in $[\![\psi_1]\!]$ with a successor state that is marked
  - repeat previous line until no change
  - $[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$ is set of marked states
- More formally: $[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!] = \bigcup_{n=0}^{\infty}[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_n$ where:

$$\begin{array}{lcl}
[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_0 & = & [\![\psi_2]\!] \\
[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_{n+1} & = & [\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_n \\
 & & \cup \\
 & & \{s \in [\![\psi_1]\!] \mid \exists s' \in [\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]_n.\ R\ s\ s'\}
\end{array}$$

- $[\![\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$ similar, but with a more complicated iteration
  - details omitted (see Huth and Ryan)

Example: checking **EF** p

- $\mathbf{EF}p = \mathbf{E}[\top\ \mathbf{U}\ p]$
  - holds if ψ holds along some path

- Note $[\![\top]\!] = S$

- Let $\mathcal{S}_n = [\![\mathbf{E}[\top\ \mathbf{U}\ p]]\!]_n$ then:

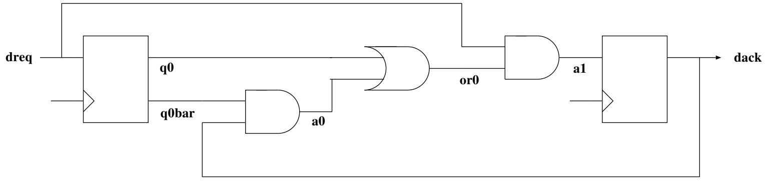
$$\begin{aligned} \mathcal{S}_0 &= [\![\mathbf{E}[\top\ \mathbf{U}\ p]]\!]_0 \\ &= [\![p]\!] \\ &= \{s \mid p \in L(s)\} \end{aligned}$$

$$\begin{aligned} \mathcal{S}_{n+1} &= \mathcal{S}_n\ \cup\ \{s \in [\![\top]\!] \mid \exists s' \in [\![\mathbf{E}[\top\ \mathbf{U}\ p]]\!]_n.\ R\ s\ s'\} \\ &= \mathcal{S}_n\ \cup\ \{s \mid \exists s' \in \mathcal{S}_n.\ R\ s\ s'\} \end{aligned}$$

  - mark all the states labelled with p
  - mark all with at least one marked successor
  - repeat until no change
  - $[\![\mathbf{EF}\ p]\!]$ is set of marked states

Example: RCV

- Recall the handshake circuit:



- State represented by a triple of Booleans $(dreq, q0, dack)$
- A model of RCV is M_{RCV} where:

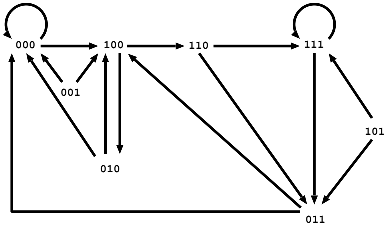
  $M = (S_{\mathrm{RCV}}, S_{0\,\mathrm{RCV}}, R_{\mathrm{RCV}}, L_{\mathrm{RCV}})$

  and

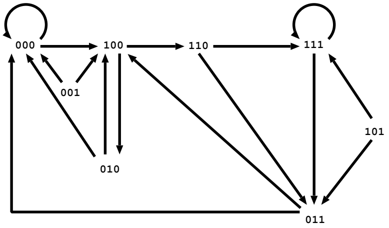
  $$\begin{aligned} R_{\mathrm{RCV}}\ (dreq, q0, dack)\ (dreq', q0', dack') = \\ (q0' = dreq) \wedge (dack' = (dreq \wedge (q0 \vee dack))) \end{aligned}$$

`RCV` as a transition system

- Possible states for `RCV`:
  $\{000, 001, 010, 011, 100, 101, 110, 111\}$
  where $b_2 b_1 b_0$ denotes state
  $\texttt{dreq} = b_2 \ \wedge\ \texttt{q0} = b_1 \ \wedge\ \texttt{dack} = b_0$
- Graph of the transition relation:



Computing $\{\mathbf{EF}\ \mathtt{At111}\}$ where $\mathtt{At111} \in L_{\mathrm{RCV}}(s) \Leftrightarrow s = 111$

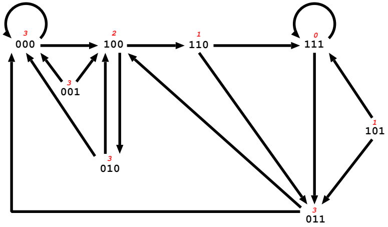


- Define:

$$\begin{aligned}
\mathcal{S}_0 &= \{s \mid \mathtt{At111} \in L_{\mathrm{RCV}}(s)\} \\
&= \{s \mid s = 111\} \\
&= \{111\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{S}_{n+1} &= \mathcal{S}_n \cup \{s \mid \exists s' \in \mathcal{S}_n.\ \mathcal{R}(s, s')\} \\
&= \mathcal{S}_n \cup \{b_2 b_1 b_0 \mid \\
&\qquad \exists b'_2 b'_1 b'_0 \in \mathcal{S}_n.\ (b'_1 = b_2) \wedge (b'_0 = b_2 \wedge (b_1 \vee b_0))\}
\end{aligned}$$

Computing $\{\!\!\{$**EF** `At111`$\}\!\!\}$ (continued)



- Compute:

$$
\begin{aligned}
\mathcal{S}_0 &= \{111\} \\
\mathcal{S}_1 &= \{111\} \cup \{101, 110\} \\
&= \{111, 101, 110\} \\
\mathcal{S}_2 &= \{111, 101, 110\} \cup \{100\} \\
&= \{111, 101, 110, 100\} \\
\mathcal{S}_3 &= \{111, 101, 110, 100\} \cup \{000, 001, 010, 011\} \\
&= \{111, 101, 110, 100, 000, 001, 010, 011\} \\
\mathcal{S}_n &= \mathcal{S}_3 \quad (n > 3)
\end{aligned}
$$

- $\{\!\!\{$**EF** `At111`$\}\!\!\}$ $= \mathbb{B}^3 = S_{\mathtt{RCV}}$
- $M_{\mathtt{RCV}} \models$ **EF** `At111` $\Leftrightarrow S_{0\,\mathtt{RCV}} \subseteq S$

Symbolic model checking

- Represent sets of states with BDDs
- Represent Transition relation with a BDD
- If BDDs of $[\![\psi]\!]$, $[\![\psi_1]\!]$, $[\![\psi_2]\!]$ are known, then:
  - BDDs of $[\![\neg\psi]\!]$, $[\![\psi_1 \wedge \psi_2]\!]$, $[\![\psi_1 \vee \psi_2]\!]$, $[\![\psi_1 \Rightarrow \psi_2]\!]$ computed using standard BDD algorithms
  - BDDs of $[\![\mathbf{AX}\psi]\!]$, $[\![\mathbf{EX}\psi]\!]$, $[\![\mathbf{A}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$, $[\![\mathbf{E}[\psi_1\ \mathbf{U}\ \psi_2]]\!]$ computed using straightforward algorithms (see textbooks)
- Model checking CTL generalises reachable-states iteration

History of Model checking

- CTL model checking due to Emerson, Clarke & Sifakis
- Symbolic model checking due to several people:
  - Clarke & McMillan (idea usually credited to McMillan's PhD)
  - Coudert, Berthet & Madre
  - Pixley
- SMV (McMillan) is a popular symbolic model checker:

`http://www.cs.cmu.edu/~modelcheck/smv.html`	(original)
`http://www.kenmcmil.com/smv.html`	(Cadence extension by McMillan)
`http://nusmv.fbk.eu/`	(new implementation)

- Other temporal logics
  - CTL*: combines CTL and LTL
  - Engineer friendly industrial languages: PSL, SVA

Expressibility of CTL

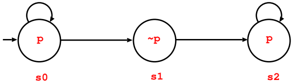
- Consider the property

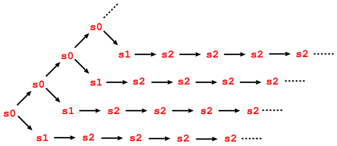
  "on every path there is a point after which p is always true on that path"

- Consider

  (($\star$) non-deterministically chooses T or F)

```
       0:   P:=1;
  s0   1:   WHILE (⋆) DO SKIP;
  s1   2:   P:=0;
  s2   3:   P:=1;
       4:   WHILE T DO SKIP;
       5:
```





- Property true, but cannot be expressed in CTL
  - would need something like **AF**ψ
  - where ψ is something like "property p *true from now on*"
  - but in CTL ψ must start with a path quantifier **A** or **E**
  - cannot talk about current path, only about all or some paths
  - **AF(AG p)** is false (consider path s0 s0 s0 $\cdots$)

LTL can express things CTL can't

- Recall:
$$\begin{aligned} [\![\mathbf{F}\phi]\!]_M(\pi) &= \exists i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \\ [\![\mathbf{G}\phi]\!]_M(\pi) &= \forall i.\ [\![\phi]\!]_M(\pi{\downarrow}i) \end{aligned}$$

- $\mathbf{FG}\phi$ is true if there is a point after which ϕ is always true
$$\begin{aligned} [\![\mathbf{FG}\phi]\!]_M(\pi) &= [\![\mathbf{F}(\mathbf{G}(\phi))]\!]_M(\pi) \\ &= \exists m_1.\ [\![\mathbf{G}(\phi)]\!]_M(\pi{\downarrow}m_1) \\ &= \exists m_1.\ \forall m_2.\ [\![\phi]\!]_M((\pi{\downarrow}m_1){\downarrow}m_2) \\ &= \exists m_1.\ \forall m_2.\ [\![\phi]\!]_M(\pi{\downarrow}(m_1+m_2)) \end{aligned}$$

- LTL can express things that CTL can't express

- Note: it's tricky to prove CTL can't express $\mathbf{FG}\phi$

CTL can express things that LTL can't express

- **AG**(**EF** p) says:

  "from every state it is possible to get to a state for which p holds"

- Can't say this in LTL (easy proof given earlier - slide 76)
- Consider disjunction:

  "on every path there is a point after which p is always true on that path
  or
  from every state it is possible to get to a state for which p holds"

- Can't say this in either CTL or LTL!
- CTL* combines CTL and LTL and can express this property

CTL*

- Both state formulae (ψ) and path formulae (ϕ)
  - state formulae ψ are true of a state *s* like CTL
  - path formulae ϕ are true of a path π like LTL
- Defined mutually recursively

$$
\begin{array}{lcll}
\psi & ::= & p & \text{(Atomic formula)} \\
 & | & \neg\psi & \text{(Negation)} \\
 & | & \psi_1 \vee \psi_2 & \text{(Disjunction)} \\
 & | & \mathbf{A}\phi & \text{(All paths)} \\
 & | & \mathbf{E}\phi & \text{(Some paths)} \\
\\
\phi & ::= & \psi & \text{(Every state formula is a path formula)} \\
 & | & \neg\phi & \text{(Negation)} \\
 & | & \phi_1 \vee \phi_2 & \text{(Disjunction)} \\
 & | & \mathbf{X}\phi & \text{(Successor)} \\
 & | & \mathbf{F}\phi & \text{(Sometimes)} \\
 & | & \mathbf{G}\phi & \text{(Always)} \\
 & | & [\phi_1\ \mathbf{U}\ \phi_2] & \text{(Until)}
\end{array}
$$

- CTL is CTL* with **X**, **F**, **G**, $[-\mathbf{U}-]$ preceded by **A** or **E**
- LTL consists of CTL* formulae of form $\mathbf{A}\phi$, where the only state formulae in ϕ are atomic

CTL* semantics

- Combines CTL state semantics with LTL path semantics:

$$
\begin{array}{lcl}
[\![p]\!]_M(s) & = & p \in L(s) \\
[\![\neg\psi]\!]_M(s) & = & \neg([\![\psi]\!]_M(s)) \\
[\![\psi_1 \vee \psi_2]\!]_M(s) & = & [\![\psi_1]\!]_M(s) \ \vee \ [\![\psi_2]\!]_M(s) \\
[\![\mathbf{A}\phi]\!]_M(s) & = & \forall\pi.\ \mathrm{Path}\ R\ s\ \pi \ \Rightarrow\ \phi(\pi) \\
[\![\mathbf{E}\phi]\!]_M(s) & = & \exists\pi.\ \mathrm{Path}\ R\ s\ \pi \ \wedge\ [\![\phi]\!]_M(\pi) \\
\\
[\![\psi]\!]_M(\pi) & = & [\![\psi]\!]_M(\pi(0)) \\
[\![\neg\phi]\!]_M(\pi) & = & \neg([\![\phi]\!]_M(\pi)) \\
[\![\phi_1 \vee \phi_2]\!]_M(\pi) & = & [\![\phi_1]\!]_M(\pi) \ \vee \ [\![\phi_2]\!]_M(\pi) \\
[\![\mathbf{X}\phi]\!]_M(\pi) & = & [\![\phi]\!]_M(\pi{\downarrow}1) \\
[\![\mathbf{F}\phi]\!]_M(\pi) & = & \exists m.\ [\![\phi]\!]_M(\pi{\downarrow}m) \\
[\![\mathbf{G}\phi]\!]_M(\pi) & = & \forall m.\ [\![\phi]\!]_M(\pi{\downarrow}m) \\
[\![[\phi_1\ \mathbf{U}\ \phi_2]]\!]_M(\pi) & = & \exists i.\ [\![\phi_2]\!]_M(\pi{\downarrow}i) \wedge \forall j.\ j{<}i \ \Rightarrow\ [\![\phi_1]\!]_M(\pi{\downarrow}j)
\end{array}
$$

- Note $[\![\psi]\!]_M : S{\to}\mathbb{B}$ and $[\![\phi]\!]_M : (\mathbb{N}{\to}S){\to}\mathbb{B}$

LTL and CTL as CTL*

- As usual: $M = (S, S_0, R, L)$
- If ψ is a CTL* state formula: $M \models \psi \;\Leftrightarrow\; \forall s \in S_0.\ [\![\psi]\!]_M(s)$
- If ϕ is an LTL path formula then: $M \models_{\text{LTL}} \phi \;\Leftrightarrow\; M \models_{\text{CTL*}} \mathbf{A}\phi$
- If R is left-total ($\forall s.\ \exists s'.\ R\ s\ s'$) then (exercise):
  $\forall s\ s'.\ R\ s\ s' \;\Leftrightarrow\; \exists\pi.\ \text{Path}\ R\ s\ \pi \wedge (\pi(1) = s')$
- The meanings of CTL formulae are the same in CTL*

$$
\begin{array}{lll}
[\![\mathbf{A}(\mathbf{X}\psi)]\!]_M(s) & & \\
= \forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow [\![\mathbf{X}\psi]\!]_M(\pi) & & \\
= \forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow [\![\psi]\!]_M(\pi{\downarrow}1) & & (\psi \text{ as path formula}) \\
= \forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow [\![\psi]\!]_M((\pi{\downarrow}1)(0)) & & (\psi \text{ as state formula}) \\
= \forall\pi.\ \text{Path}\ R\ s\ \pi \Rightarrow [\![\psi]\!]_M(\pi(1)) & &
\end{array}
$$

$$
\begin{array}{l}
[\![\mathbf{AX}\psi]\!]_M(s) \\
= \forall s'.\ R\ s\ s' \;\Rightarrow\; [\![\psi]\!]_M(s') \\
= \forall s'.\ (\exists\pi.\ \text{Path}\ R\ s\ \pi \wedge (\pi(1) = s')) \;\Rightarrow\; [\![\psi]\!]_M(s') \\
= \forall s'.\ \forall\pi.\ \text{Path}\ R\ s\ \pi \wedge (\pi(1) = s') \;\Rightarrow\; [\![\psi]\!]_M(s') \\
= \forall\pi.\ \text{Path}\ R\ s\ \pi \;\Rightarrow\; [\![\psi]\!]_M(\pi(1))
\end{array}
$$

Exercise: do similar proofs for other CTL formulae

Fairness

- May want to assume system or environment is ‘fair’
- Example 1: fair arbiter
  the arbiter doesn’t ignore one of its requests forever
  - not every request need be granted
  - want to exclude infinite number of requests and no grant
- Example 2: reliable channel
  no message continuously transmitted but never received
  - not every message need be received
  - want to exclude an infinite number of sends and no receive

Handling fairness in CTL and LTL

- Consider:
  p holds infinitely often along a path then so does q
- In LTL is expressible as $\mathbf{G}(\mathbf{F}\ p) \ \Rightarrow\ \mathbf{G}(\mathbf{F}\ q)$
- Can't say this in CTL
  - why not – what's wrong with $\mathbf{AG}(\mathbf{AF}\ p) \ \Rightarrow\ \mathbf{AG}(\mathbf{AF}\ q)$?
  - in CTL* expressible as $\mathbf{A}\Big(\mathbf{G}(\mathbf{F}\ p) \ \Rightarrow\ \mathbf{G}(\mathbf{F}\ q)\Big)$
  - fair CTL model checking implemented in checking algorithm
  - fair LTL just a fairness assumption like $\mathbf{G}(\mathbf{F}\ p) \ \Rightarrow\ \cdots$
- Fairness is a tricky and subtle subject
  - many kinds of fairness:
    'weak fairness', 'strong fairness' etc
  - exist whole books on fairness

TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE

FAIRNESS

Nissim Francez

SPRINGER-VERLAG
NEW YORK BERLIN HEIDELBERG TOKYO

Richer Logics than LTL and CTL

- Propositional modal μ-calculus
- Industrial Languages, e.g. PSL
- Modal Logics, where modes can be other than time in temporal logic. Examples:
  - Logics including possibility and necessity
  - Logics of belief: “P believes that Q believes F”
  - Logics of authentication, e.g. BAN logic

  More information can be found under “Modal Logic", “Doxastic logic” and “Burrows-Abadi-Needham logic” on Wikipedia.

[not examinable]

Propositional modal μ-calculus

- You may learn this in *Topics in Concurrency*
- μ-calculus is an even more powerful property language
  - has fixed-point operators
  - both maximal and minimal fixed points
  - model checking consists of calculating fixed points
  - many logics (e.g. CTL*) can be translated into μ-calculus
- Strictly stronger than CTL*
  - expressibility strictly increases as allowed nesting increases
  - need fixed point operators nested 2 deep for CTL*
- The μ-calculus is very non-intuitive to use!
  - intermediate code rather than a practical property language
  - nice meta-theory and algorithms, but terrible usability!

[not examinable]

PSL/Sugar

- Used for real-life hardware verification
- Combines together LTL and CTL
- SEREs: Sequential Extended Regular Expressions
- LTL – Foundation Language formulae
- CTL – Optional Branching Extension
- Relatively simple set of primitives + definitional extension
- Boolean, temporal, verification, modelling layers
- Semantics for static and dynamic verification
  (needs strong/weak distinction)
- You may learn more about this in *System-on-Chip Design*

[not examinable]

Bisimulation equivalence: general idea

- M, M' bisimilar if they have ‘corresponding executions’
  - to each step of M there is a corresponding step of M'
  - to each step of M' there is a corresponding step of M
- Bisimilar models satisfy same CTL* properties
- Bisimilar: same truth/falsity of model properties
- Simulation gives property-truth preserving abstraction (see later)

Bisimulation relations

- Let $R : S \to S \to \mathbb{B}$ and $R' : S' \to S' \to \mathbb{B}$ be transition relations
- B is a bisimulation relation between R and R' if:
  - $B : S \to S' \to \mathbb{B}$
  - $\forall s\ s'.\ B\ s\ s' \Rightarrow \forall s_1 \in S.\ R\ s\ s_1 \Rightarrow \exists s_1'.\ R'\ s'\ s_1' \wedge B\ s_1\ s_1'$
    (to each step of R there is a corresponding step of R')
  - $\forall s\ s'.\ B\ s\ s' \Rightarrow \forall s_1' \in S.\ R'\ s'\ s_1' \Rightarrow \exists s_1.\ R\ s\ s_1 \wedge B\ s_1\ s_1'$
    (to each step of R' there is a corresponding step of R)

Bisimulation equivalence: definition and theorem

- Let $M = (S, S_0, R, L)$ and $M' = (S', S'_0, R', L')$
- $M \equiv M'$ if:
  - there is a bisimulation B between R and R'
  - $\forall s_0 \in S_0.\ \exists s'_0 \in S'_0.\ B\ s_0\ s'_0$
  - $\forall s'_0 \in S'_0.\ \exists s_0 \in S_0.\ B\ s_0\ s'_0$
  - $\forall s\ s'.\ B\ s\ s' \Rightarrow L(s) = L'(s')$
- Theorem: if $M \equiv M'$ then for any CTL* state formula ψ:
  $M \models \psi \ \Leftrightarrow\ M' \models \psi$
- See Q14 in the Exercises

Abstraction and Abstraction Refinement

Abstraction

- Abstraction creates a simplification of a model
  - separate states/atomic properties may get merged
  - an abstract path can represent several concrete paths
- $M \preceq \overline{M}$ means $\overline{M}$ is an abstraction of M
  - to each step of M there is a corresponding step of $\overline{M}$
  - atomic properties of M correspond to atomic properties of $\overline{M}$
- Special case (close to bisimulation) is when $\overline{M}$ is a subset of M such that:
  - $\overline{M} = (\overline{S_0}, \overline{S}, \overline{R}, \overline{L})$ and $M = (S_0, S, R, L)$
    $\overline{S} \subseteq S$
    $\overline{S_0} = S_0$
    $\forall s\ s' \in \overline{S}.\ \overline{R}\ s\ s' \Leftrightarrow R\ s\ s'$
    $\forall s \in \overline{S}.\ \overline{L}\ s = L\ s$
  - $\overline{S}$ contain all reachable states of M
    $\forall s \in \overline{S}.\ \forall s' \in S.\ R\ s\ s' \Rightarrow s' \in \overline{S}$
- All paths of M from initial states are $\overline{M}$-paths
  - hence for all CTL formulae ψ: $\overline{M} \models \psi \ \Rightarrow M \models \psi$

Recall JM1

```
Thread 1                              Thread 2
0:  IF LOCK=0 THEN LOCK:=1;           0:  IF LOCK=0 THEN LOCK:=1;
1:  X:=1;                             1:  X:=2;
2:  IF LOCK=1 THEN LOCK:=0;           2:  IF LOCK=1 THEN LOCK:=0;
3:                                    3:
```

- Two program counters, state: $(pc_1, pc_2, lock, x)$

$$S_{\texttt{JM1}} = [0..3] \times [0..3] \times \mathbb{Z} \times \mathbb{Z}$$

$$\begin{array}{lll|lll}
R_{\texttt{JM1}} & (0, pc_2, 0, x) & (1, pc_2, 1, x) & R_{\texttt{JM1}} & (pc_1, 0, 0, x) & (pc_1, 1, 1, x) \\
R_{\texttt{JM1}} & (1, pc_2, lock, x) & (2, pc_2, lock, 1) & R_{\texttt{JM1}} & (pc_1, 1, lock, x) & (pc_1, 2, lock, 2) \\
R_{\texttt{JM1}} & (2, pc_2, 1, x) & (3, pc_2, 0, x) & R_{\texttt{JM1}} & (pc_1, 2, 1, x) & (pc_1, 3, 0, x)
\end{array}$$

- Assume $\texttt{NotAt11} \in L_{\texttt{JM1}}(pc_1, pc_2, lock, x) \Leftrightarrow \neg((pc_1 = 1) \wedge (pc_2 = 1))$
- Model $M_{\texttt{JM1}} = (S_{\texttt{JM1}}, \{(0,0,0,0)\}, R_{\texttt{JM1}}, L_{\texttt{JM1}})$
- $S_{\texttt{JM1}}$ not finite, but actually $lock \in \{0,1\}$, $x \in \{0,1,2\}$
- Clear by inspection that $M_{\texttt{JM1}} \preceq \overline{M}_{\texttt{JM1}}$ where:

  $\overline{M}_{\texttt{JM1}} = (\overline{S}_{\texttt{JM1}}, \{(0,0,0,0)\}, \overline{R}_{\texttt{JM1}}, \overline{L}_{\texttt{JM1}})$

  - $\overline{S}_{\texttt{JM1}} = [0..3] \times [0..3] \times [0..1] \times [0..2]$
  - $\overline{R}_{\texttt{JM1}}$ is $R_{\texttt{JM1}}$ restricted to arguments from $\overline{S}_{\texttt{JM1}}$
  - $\texttt{NotAt11} \in \overline{L}_{\texttt{JM1}}(pc_1, pc_2, lock, x) \Leftrightarrow \neg((pc_1 = 1) \wedge (pc_2 = 1))$
  - $\overline{L}_{\texttt{JM1}}$ is $L_{\texttt{JM1}}$ restricted to arguments from $\overline{S}_{\texttt{JM1}}$

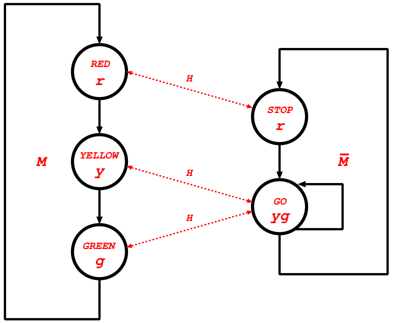
Simulation relations

- Let $R : S \to S \to \mathbb{B}$ and $\overline{R} : \overline{S} \to \overline{S} \to \mathbb{B}$ be transition relations
- H is a simulation relation between R and $\overline{R}$ if:
  - H is a relation between S and $\overline{S}$ – i.e. $H : S \to \overline{S} \to \mathbb{B}$
  - to each step of R there is a corresponding step of $\overline{R}$ – i.e.: $\forall s\ \overline{s}.\ H\ s\ \overline{s} \Rightarrow \forall s' \in S.\ R\ s\ s' \Rightarrow \exists \overline{s'} \in \overline{S}.\ \overline{R}\ \overline{s}\ \overline{s'} \wedge H\ s'\ \overline{s'}$
- Also need to consider abstraction of atomic properties
  - $H_{AP} : AP \to \overline{AP} \to \mathbb{B}$
  - details glossed over here

Simulation preorder: definition and theorem

- Given two models $M = (S, S_0, R, L)$ and $\overline{M} = (\overline{S}, \overline{S_0}, \overline{R}, \overline{L})$ we say M abstracts $\overline{M}$, written $M \preceq \overline{M}$, if:
  - there is a simulation H between R and $\overline{R}$
  - $\forall s_0 \in S_0.\ \exists \overline{s_0} \in \overline{S_0}.\ H\ s_0\ \overline{s_0}$
  - $\forall s\ \overline{s}.\ H\ s\ \overline{s} \Rightarrow H_{AP}\ L(s)\ \overline{L}(\overline{s})$
- We *define* ACTL to be the subset of CTL without **E**-properties and with negation only applied to atomic properties.
  - e.g. **AG AF**p – from anywhere can always reach a p-state
  - useful for abstraction:
- Theorem: if $M \preceq \overline{M}$ then for any ACTL state formula ψ: $\overline{M} \models \psi \Rightarrow M \models \psi$
- BUT: if $\overline{M} \models \psi$ fails then cannot conclude $M \models \psi$ false
- Like abstract interpretation in Optimising Compilers

Example (Grumberg)



H a simulation

$H\ RED\ STOP \quad \wedge$
$H\ YELLOW\ GO \quad \wedge$
$H\ GREEN\ GO$

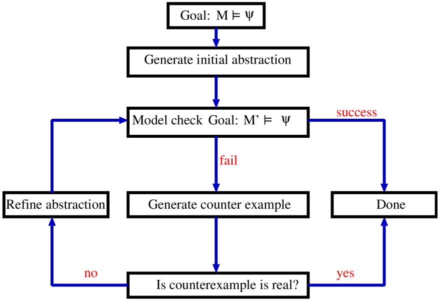
$H_{AP} : \{r, y, g\} \rightarrow \{r, yg\} \rightarrow \mathbb{B}$

$H_{AP}\ r\ r\ \wedge$
$H_{AP}\ y\ yg\ \wedge$
$H_{AP}\ g\ yg$

- $\overline{M} \models \mathbf{AG\ AF}\ \neg r$ hence $M \models \mathbf{AG\ AF}\ \neg r$
- but $\neg(\overline{M} \models \mathbf{AG\ AF}\ r)$ doesn't entail $\neg(M \models \mathbf{AG\ AF}\ r)$
  - $[\![\mathbf{AG\ AF}\ r]\!]_{\overline{M}}(STOP)$ is false
    (consider $\overline{M}$-path π' where $\pi' = STOP.GO.GO.GO.\cdots$)
  - $[\![\mathbf{AG\ AF}\ r]\!]_{M}(RED)$ is true
    (abstract path π' doesn't correspond to a real path in M)

CEGAR

- Counter Example Guided Abstraction Refinement



- Lots of details to fill out (several different solutions)
  - how to generate abstraction
  - how to check counterexamples
  - how to refine abstractions
- Microsoft SLAM driver verifier is a CEGAR system

[not examinable]

Temporal Logic and Model Checking – Summary

- Various property languages: LTL, CTL, PSL (Prior, Pnueli)
- Kripke models abstract from hardware or software designs
- Model checking checks $M \models \psi$ (Clarke et al.)
- Symbolic model checking uses BDDs (McMillan)
- Avoid state explosion via simulation and abstraction
- CEGAR refines abstractions by analysing counterexamples
- Triumph of application of computer science theory
  - two Turing awards, McMillan gets 2010 CAV award
  - widespread applications in industry

THE END