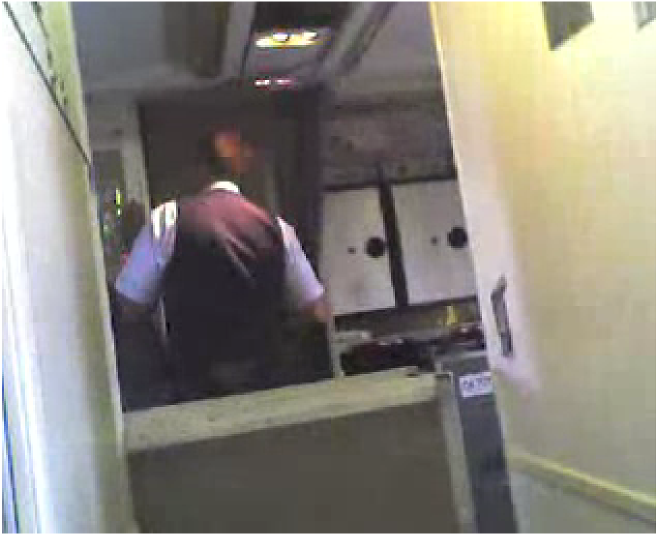
Verification of Real Time Systems - CS5270
9th lecture

Hugh Anderson

National University of Singapore
School of Computing

March, 2007

Plane comfort systems failure...



Outline

1. Administration
   - Assignment 2
   - The road map...
2. More preliminaries for model checking
   - Model checking setting
   - The Kripke structure...
3. CTL model checking
   - CTL formulæ
   - Semantics of CTL - the modelling relation
   - The model checking algorithm

Assignment 2

A reminder... Assignment number 2:

- On the web site
- Due next week! ...

The immediate road map

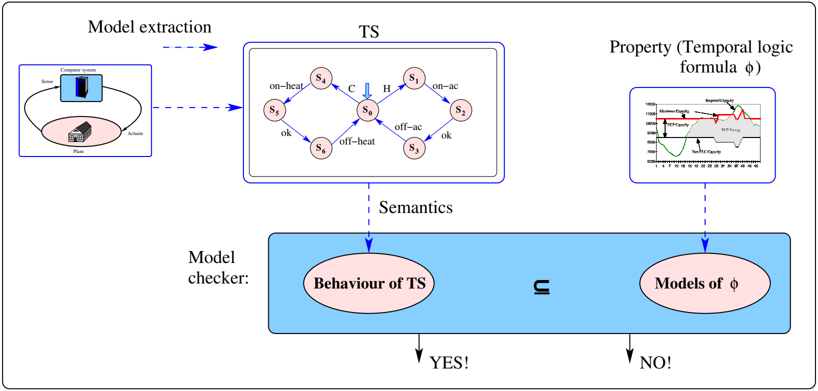
The topics:

- **TTS: Timed transition systems**
  - Reduction: $\mathrm{TTS} \rightarrow \mathrm{TS}_{\mathrm{TTS}} \rightarrow \mathrm{TA}_{\mathrm{TTS}} \rightarrow \mathrm{QTS}$, regions and zones (zone operations, DBMs)
- **Preliminaries for Model Checking**
  - Behaviour, safety, liveness, automata, reachability

  - Temporal logic
  - Foundations for CTL/TCTL model checking (Kripke semantics)
- **Model Checking**
  - The model checking relation
  - The model checking algorithm, with optimizations

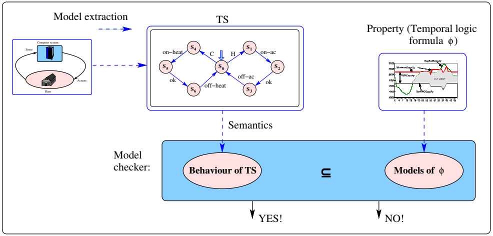
The big picture...

Properties and behaviour:



The big picture...

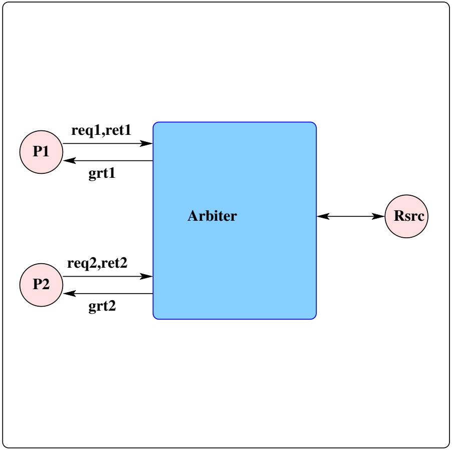
Properties and behaviour:



- TS represents the behaviour of the system, expressed as the allowable set of runs (or computations) of the system.
- A model-checker checks if this *behaviour* of the system is a subset of the set of runs (or computations) induced by an arbitrary property ϕ, returning YES or NO.

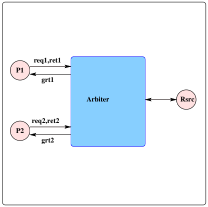
A simple system

Resource arbiter:



A simple system

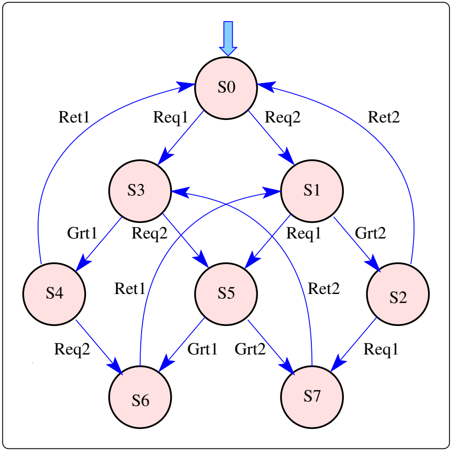
Resource arbiter:



- Arbiter: allows one process at a time to access resource.
- Process: requests access to resource, by `req()` call.
- When resource is free, arbiter grants access by signalling the process using `grt()` signal.
- Process: no longer needs resource, signals arbiter: `ret()`.

Model behaviour of simple system

Resource arbiter transition system:



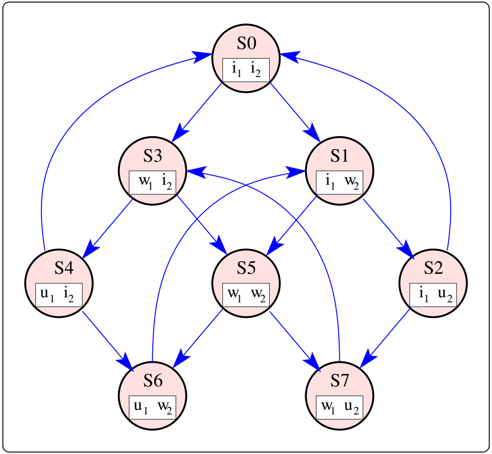
Properties for simple system

Atomic propositions for system:

- Important to identify suitable atomic propositions relevant to the system. Suitable propositions might be:
  - i_1, i_2 : Processes 1 and 2 are *idle*. In the starting state both processes are idle.
  - w_1, w_2 : Processes 1 and 2 are *waiting* for the resource.
  - u_1, u_2 : Processes 1 and 2 are *using* the resource.

Labelling the system...

Add atomic propositions, remove actions...



Kripke semantics and structures

A formal semantics for modal logic systems:

The $\Box$ operator cannot be formalized with an extensional semantics. Kripke semantics is a formal semantics for modal logic systems. It is defined over a Kripke frame/model/structure:

Definition: A Kripke structure $\mathcal{K}$ over a set AP of atomic propositions is a 4-tuple $(S, \Delta, AP, \mathcal{L})$, where

- S is a finite set of **states**
- $\Delta \subseteq S \times S$ is a **transition relation** that must be *total*
- AP is a finite set of **atomic propositions**
- $\mathcal{L} : S \rightarrow 2^{AP}$ is a function which labels each state with the set of atomic propositions true in that state

Example Kripke structure

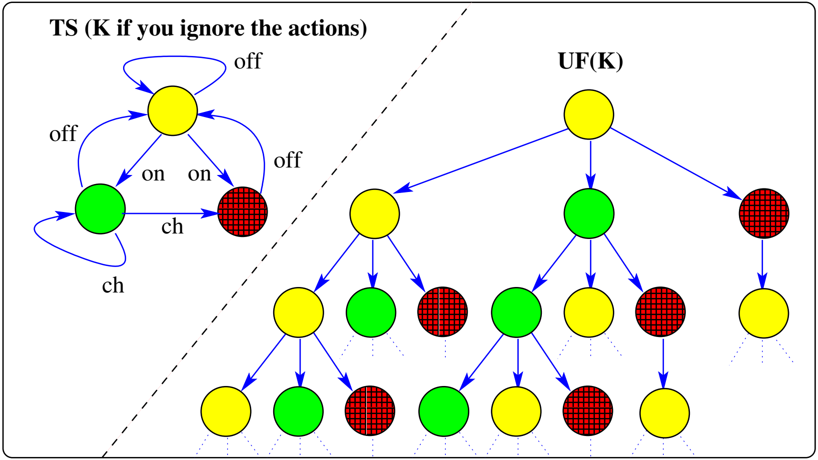
The arbiter system:

- we have $AP = \{i_1, w_1, u_1, i_2, w_2, u_2\}$
- Write out $\mathcal{L}(s)$ for each state s. The labelling function $\mathcal{L} : S \to 2^{\mathrm{AP}}$:

$$
\begin{aligned}
\mathcal{L} = \{ \quad & (s_0, \{i_1, i_2\}), \\
& (s_1, \{i_1, w_2\}), \\
& (s_2, \{i_1, u_2\}), \\
& (s_3, \{w_1, i_2\}), \\
& (s_4, \{u_1, i_2\}), \\
& (s_5, \{w_1, w_2\}), \\
& (s_6, \{u_1, w_2\}), \\
& (s_7, \{w_1, u_2\}) \quad \}
\end{aligned}
$$

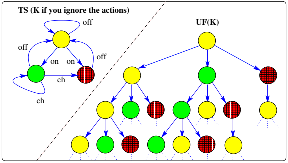
Unfolding the Kripke structure

Easier to visualize UF(K):



Unfolding the Kripke structure

Definition:



$\mathrm{UF}(\mathcal{K})$ is another Kripke structure.

Definition: The **unfolding** of a Kripke structure $\mathcal{K}$, from an identified starting state s_0, is $\mathrm{UF}(\mathcal{K}) = (S, \Delta, AP, \mathcal{L})$, where

- $S = \{(s, \pi) \mid \pi \text{ is a path from } s_0 \text{ to } s \text{ in } \mathcal{K}$
- $\Delta((s, \pi), (s', \pi')) \text{ iff } \Delta(s, s') \text{ in } \mathcal{K} \text{ and } \pi' = \pi s'$.
- $\mathcal{L}(s, \pi) = \mathcal{L}(s)$

CTL formulæ

The form:

- In CTL formulæ each of the temporal operators must be preceded by a *path* quantifier: A, or E.
- There are ten base expressions as a result, but we only actually need 3 expressions:
    - $\mathrm{EX}\,p$: For one computation path, property p holds in the next state;
    - $\mathrm{A}(p\,\mathrm{U}\,q)$: For all computation paths, property p holds until q holds.
    - $\mathrm{E}(p\,\mathrm{U}\,q)$: For one computation path, property p holds until q holds.
- (Call this CTL-)

CTL- formulæ

Definition for CTL-

Given a proposition $p \in AP$ (a finite set of atomic propositions), then p is a CTL- formula, and if ψ_1 and ψ_2 are CTL- formulæ, then

- $\neg\psi_1$ is a CTL- formula
- $\psi_1 \wedge \psi_2$ is a CTL- formula
- $\psi_1 \vee \psi_2$ is a CTL- formula
- $\mathrm{EX}(\psi_1)$ is a CTL- formula
- $\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2)$ is a CTL- formula
- $\mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2)$ is a CTL- formula

Semantics of CTL-

Expressed in terms of model and modelling relation...

- Model checking is commonly expressed as a ternary relation $(\models)$:

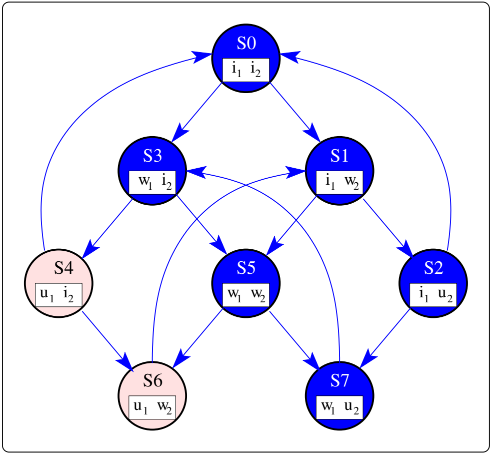
$$M, s \models P$$

- The relation is true when the property P holds in state s for a given model M.

- It is normally defined inductively, with a set of interlocking rules.
- A labelling algorithm may then be used to establish the set of states satisfying the relation.

Labelling the system for $\mathrm{EX}(w_1)$...

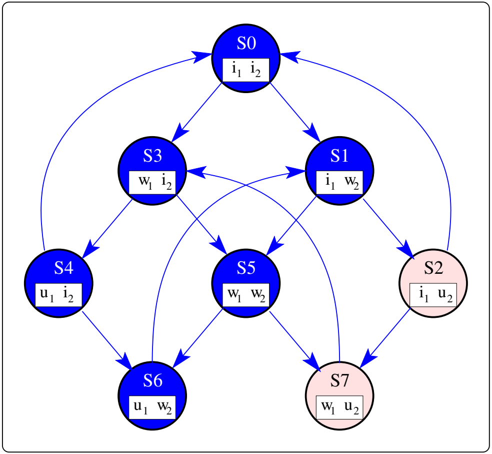
States coloured blue have desired temporal formula...



$M, s_0 \models \mathrm{EX}(w_1)$?

Labelling the system for $\mathrm{E}\,(i_2\,\mathrm{U}\,w_2)$...

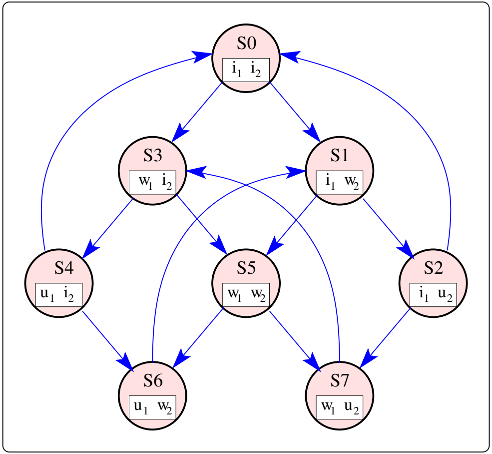
States coloured blue have desired temporal formula...



$M, s_2 \models \mathrm{E}(i_2\,\mathrm{U}\,w_2)$?

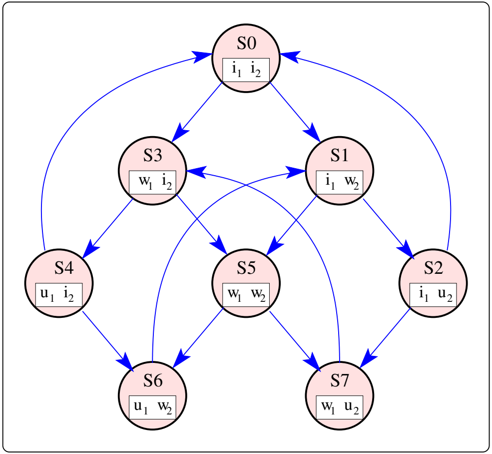
$M, s_2 \models \mathrm{E}(u_2 \,\mathrm{U}\, w_1)$?

Label states, check inclusion...



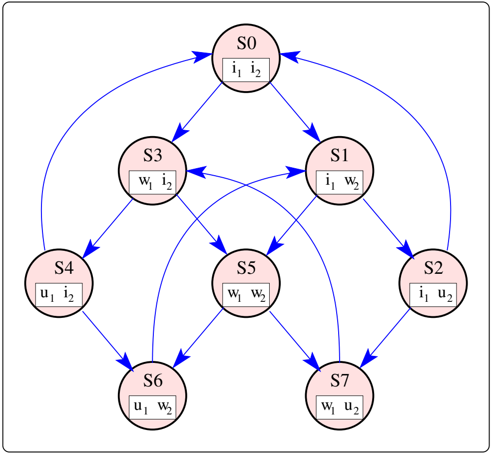
$M, s_2 \models \mathrm{A}(u_2\,\mathrm{U}\,w_1)$?

Label states, check inclusion...



$M, s_2 \models \mathrm{A}(u_2 \,\mathrm{U}\, i_2)$?

Label states, check inclusion...



Inductive definition of the modelling relation

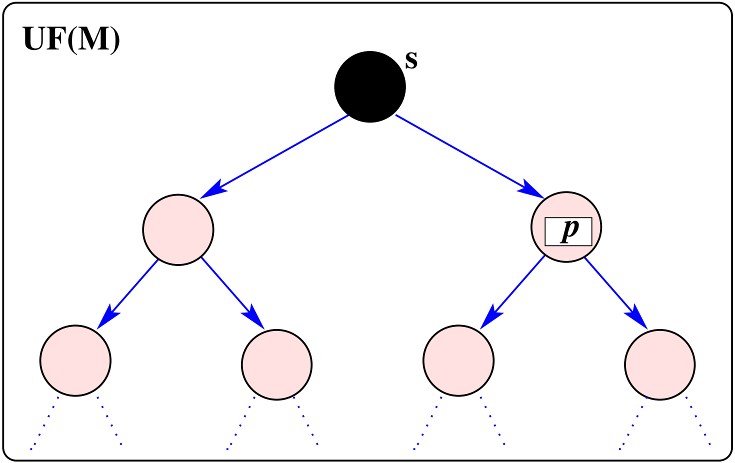
The model checking relation is defined for...

...each atomic proposition p and each CTL- formula ψ_1, ψ_2 as:

$M, s \models p$	$\Leftrightarrow$	$p \in \mathcal{L}(s)$
$M, s \models \neg\psi_1$	$\Leftrightarrow$	iff it is not the case that $M, s \models \psi_1$
$M, s \models \psi_1 \wedge \psi_2$	$\Leftrightarrow$	iff $M, s \models \psi_1$ and $M, s \models \psi_2$
$M, s \models \psi_1 \vee \psi_2$	$\Leftrightarrow$	iff $M, s \models \psi_1$ or $M, s \models \psi_2$
$M, s \models \mathrm{EX}(\psi_1)$	$\Leftrightarrow$	iff $\Delta(s, s')$ and $M, s' \models \psi_1$ (i.e. s has a successor state at which ψ_1 holds)
$M, s \models \mathrm{A}(\psi_1 \, \mathrm{U} \, \psi_2)$	$\Leftrightarrow$	iff for every path $\pi = s_0 \; s_1 \; \ldots$ from s, for some j, $M, \pi(j) \models \psi_2$, and $\forall i < j \; M, \pi(i) \models \psi_1$
$M, s \models \mathrm{E}(\psi_1 \, \mathrm{U} \, \psi_2)$	$\Leftrightarrow$	iff there is a path $\pi = s_0 \; s_1 \; \ldots$ from s, where for some j, $M, \pi(j) \models \psi_2$, and $\forall i < j \; M, \pi(i) \models \psi_1$

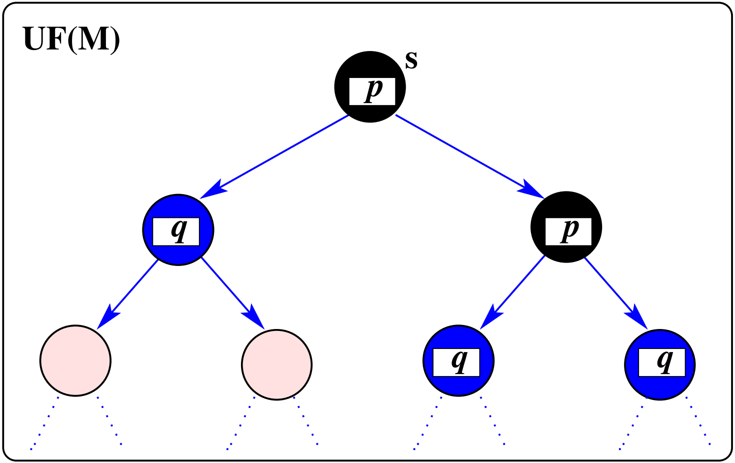
Temporal CTL operator $M, s \models \mathrm{EX}(p)$ in UF(K)

Easier to see when unfolded:



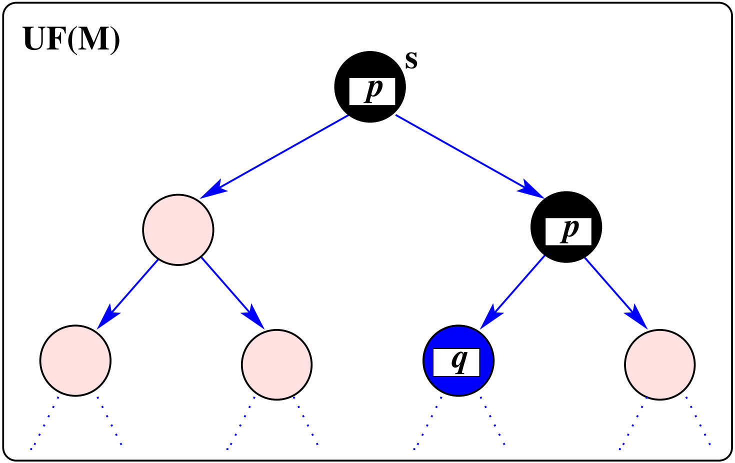
Temporal CTL operator $M, s \models \mathrm{A}(p\,\mathrm{U}\,q)$ in UF(K)

Easier to see when unfolded:



Temporal CTL operator $M, s \models \mathrm{E}(p \, \mathrm{U} \, q)$ in UF(K)

Easier to see when unfolded:



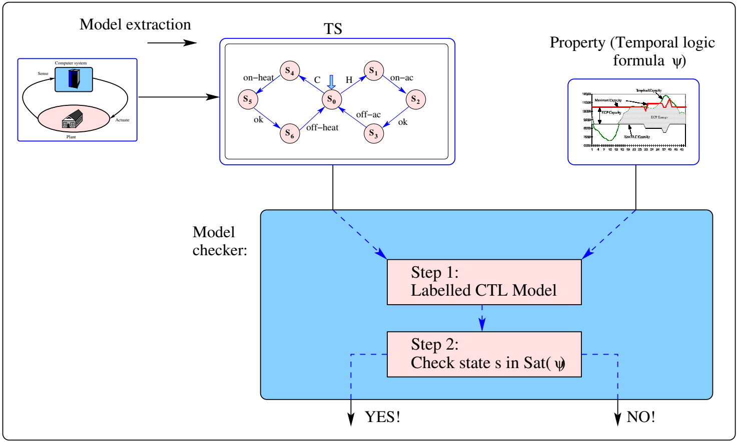
Defining CTL operators in CTL-

Two of the missing operators:

- $\mathrm{AX}(\psi) = \neg\mathrm{EX}(\neg\psi)$ For every next state ψ holds. It is not the case that there exists a next state at which ψ does not hold.
- $\mathrm{EG}(\psi) = \neg\mathrm{A}(\mathrm{true}\,\mathrm{U}\,\neg\psi)$ There exists a path π from s such that for every $k \geq 0$: $M, \pi(k) \models \psi$. It is not the case that ...

The model checking process for $M, s \models \psi$

Label states, check inclusion:



The satisfaction function for CTL model checking

Returns a set of states:

set_of_States sat(Property ψ**) =**

  if $\psi \in AP$ **then** $\{s \mid \psi \in \mathcal{L}(s)\}$

  else case ψ **of**

true:	S
false:	$\emptyset$
$\neg\psi$**:**	$S - \mathbf{sat}(\psi)$
$\psi_1 \wedge \psi_2$**:**	$\mathbf{sat}(\psi_1) \cap \mathbf{sat}(\psi_2)$
$\psi_1 \vee \psi_2$**:**	$\mathbf{sat}(\psi_1) \cup \mathbf{sat}(\psi_2)$
$\mathbf{EX}(\psi_1)$**:**	$\{s \in S \mid s' \in s^{\uparrow} \wedge s' \in \mathbf{sat}(\psi_1)\}$
$\mathbf{A}(\psi_1\, \mathbf{U}\, \psi_2)$**:**	$\mathbf{lfp}(g(Z) = \mathbf{sat}(\psi_2) \cup (\mathbf{sat}\ (\psi_1) \cap \{s \in S \mid \forall s' \in s^{\uparrow} \cap Z\}))$
$\mathbf{E}(\psi_1\, \mathbf{U}\, \psi_2)$**:**	$\mathbf{lfp}(h(Z) = \mathbf{sat}(\psi_2) \cup (\mathbf{sat}\ (\psi_1) \cap \{s \in S \mid \exists s' \in s^{\uparrow} \cap Z\}))$

The satisfaction function for CTL model checking

Least fix-point:

We can calculate the sets of states for $\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2)$ and $\mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2)$, by taking the least fix-point of functions g and h (sometimes expressed as the algorithms $\mathrm{sat}_{\mathrm{AU}}$ and $\mathrm{sat}_{\mathrm{EU}}$). What are the functions g and h? Some investigation will show that

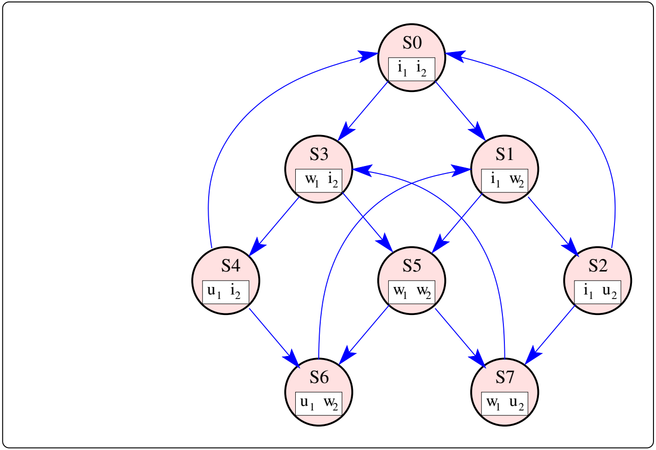
$$\begin{aligned} \mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2) &= \psi_2 \vee (\psi_1 \wedge \mathrm{AX}(\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2))), \text{ and} \\ \mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2) &= \psi_2 \vee (\psi_1 \wedge \mathrm{EX}(\mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2))) \end{aligned}$$

Express as fix-points of the corresponding functions

$$\begin{aligned} g(Z) &= \psi_2 \vee (\psi_1 \wedge \mathrm{AX}(Z)), \text{ and} \\ h(Z) &= \psi_2 \vee (\psi_1 \wedge \mathrm{EX}(Z)) \end{aligned}$$

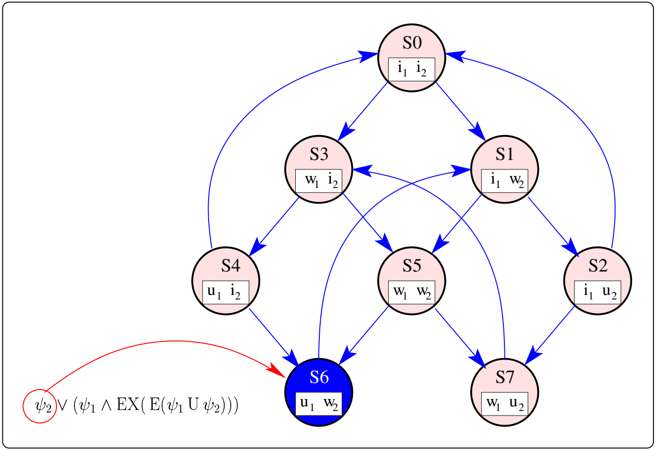
Checking $M, s_2 \models \mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2))$...

Start with the labelled Kripke structure:



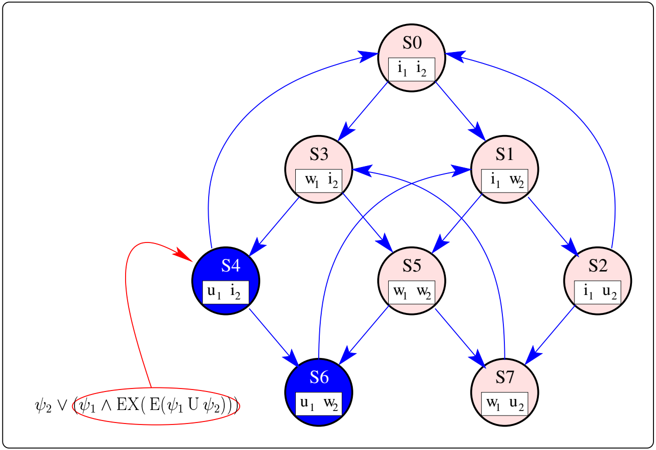
Checking $\mathrm{sat}(\mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2)))$...

Using lfp equation:



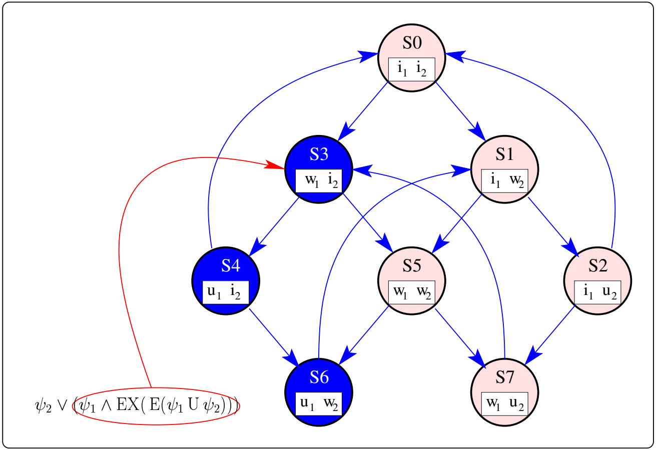
Checking $\mathrm{sat}(\mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2)))$...

Using lfp equation:



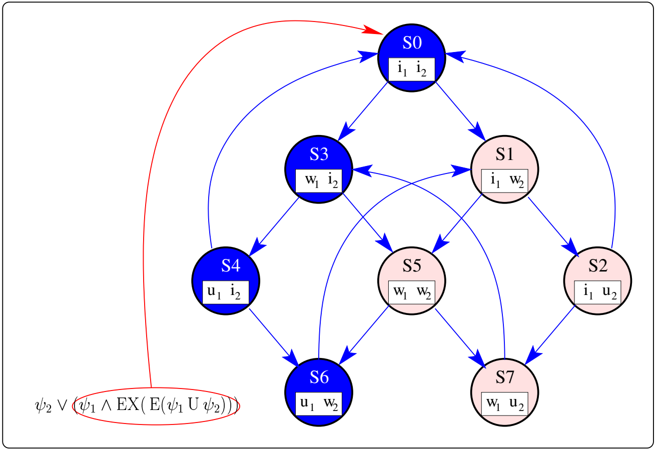
Checking $\mathrm{sat}(\mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2)))$...

Using lfp equation:



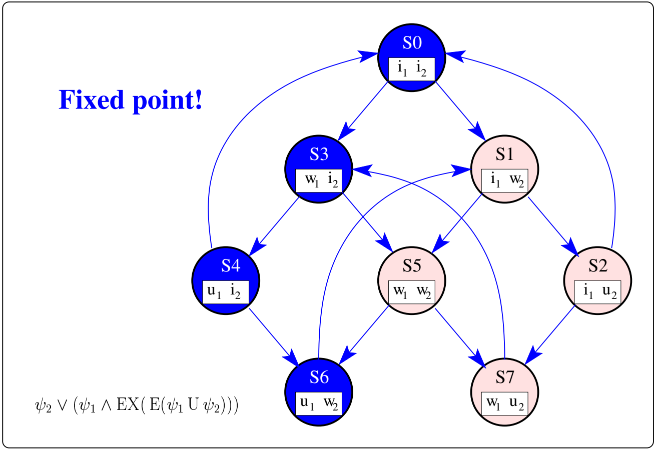
Checking $\mathrm{sat}(\mathrm{E}(i_2\,\mathrm{U}\,(u_1 \wedge w_2)))$...

Using lfp equation:



$M, s_2 \models \mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2))$... if $s_2 \in \mathrm{sat}(\mathrm{E}(i_2 \,\mathrm{U}\, (u_1 \wedge w_2)))$

Once we reach the fix-point:



Example (A *different* arbiter)

Difficult to find convincing examples that are small:

- We choose to use as an example a simple mutual exclusion protocol in which
  - two processes, P_1 and P_2 share six boolean variables, and
  - co-operate to ensure mutually exclusive access to a critical section of code.
- A third process T_1 monitors the variables and changes a turn variable.

- The entire system is the parallel composition of these three processes, and is continuous.
- Each line of code is considered to be atomic, and we use 1 to represent true, 0 to represent false.

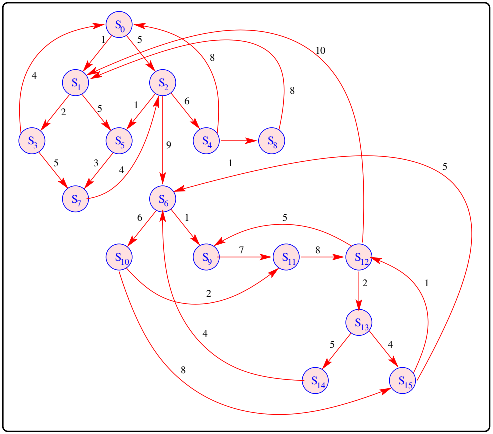
A *different* arbiter

The source code:

$$
\begin{array}{lll}
\mathrm{P}_1 & \mathbf{=} & \mathtt{\textbf{if}}\ \mathrm{idle}_1\ \mathtt{\textbf{then}}\ (\mathrm{wait}_1 := 1; \mathrm{idle}_1 := 0)\ \mathtt{\textbf{else}} \\
& & \mathtt{\textbf{if}}\ \mathrm{wait}_1 \wedge \mathrm{idle}_2\ \mathtt{\textbf{then}}\ (\mathrm{active}_1 := 1; \mathrm{wait}_1 := 0)\ \mathtt{\textbf{else}} \\
& & \mathtt{\textbf{if}}\ \mathrm{wait}_1 \wedge \mathrm{wait}_2 \wedge \neg\mathrm{turn}\ \mathtt{\textbf{then}}\ (\mathrm{active}_1 := 1; \mathrm{wait}_1 := 0)\mathtt{\textbf{;}} \\
& & \mathtt{\textbf{if}}\ \mathrm{active}_1\ \mathtt{\textbf{then}}\ (\mathrm{CritSect}; \mathrm{idle}_1 := 1; \mathrm{active}_1 := 0)\mathtt{\textbf{;}} \\
\mathrm{P}_2 & \mathbf{=} & \mathtt{\textbf{if}}\ \mathrm{idle}_2\ \mathtt{\textbf{then}}\ (\mathrm{wait}_2 := 1; \mathrm{idle}_2 := 0)\ \mathtt{\textbf{else}} \\
& & \mathtt{\textbf{if}}\ \mathrm{wait}_2 \wedge \mathrm{idle}_1\ \mathtt{\textbf{then}}\ (\mathrm{active}_2 := 1; \mathrm{wait}_2 := 0)\ \mathtt{\textbf{else}} \\
& & \mathtt{\textbf{if}}\ \mathrm{wait}_2 \wedge \mathrm{wait}_1 \wedge \mathrm{turn}\ \mathtt{\textbf{then}}\ (\mathrm{active}_2 := 1; \mathrm{wait}_2 := 0)\mathtt{\textbf{;}} \\
& & \mathtt{\textbf{if}}\ \mathrm{active}_2\ \mathtt{\textbf{then}}\ (\mathrm{CritSect}; \mathrm{idle}_2 := 1; \mathrm{active}_2 := 0)\mathtt{\textbf{;}} \\
\mathrm{T}_1 & \mathbf{=} & \mathtt{\textbf{if}}\ \mathrm{idle}_1 \wedge \mathrm{wait}_2\ \mathtt{\textbf{then}}\ \mathrm{turn} := 1\ \mathtt{\textbf{else}} \\
& & \mathtt{\textbf{if}}\ \mathrm{idle}_2 \wedge \mathrm{wait}_1\ \mathtt{\textbf{then}}\ \mathrm{turn} := 0\mathtt{\textbf{;}} \\
\mathrm{System} & \mathbf{=} & (\mathrm{P}_1 \parallel \mathrm{P}_2 \parallel \mathrm{T}_1)\mathtt{\textbf{;}}\ \ \mathrm{System}\mathtt{\textbf{;}}
\end{array}
$$

Transition diagram

Numbers relate back to program:



How do we get this?

Encoding states as boolean formulæ:

- Encode states using m boolean variables.
  - Allows for 2^m states.
  - For example: $m = 3$: $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$
- Propositional booleans a, b, c:
  - $S = \{000, 001, 010, 011, 100, 101, 110, 111\}$
  - $S = \{\neg a \wedge \neg b \wedge \neg c, \neg a \wedge \neg b \wedge c, \neg a \wedge b \wedge \neg c, \neg a \wedge b \wedge c, \ldots, a \wedge b \wedge c\}$
- Encode transitions using *before* $(a, b, c)$and *after* (a', b', c') variables.
  - For example: $(s_1, s_4) = (\neg a \wedge \neg b \wedge \neg c) \wedge (\neg a' \wedge b' \wedge c')$

Transition relation as a predicate

Transition system ends up as a boolean formula:

P_1 is $(i_1 \wedge w_1' \wedge \overline{i_1'}) \vee (w_1 \wedge i_2 \wedge a_1' \wedge \overline{w_1'}) \vee (w_1 \wedge w_2 \wedge \overline{t} \wedge a_1' \wedge \overline{w_1'}) \vee (a_1 \wedge i_1' \wedge \overline{a_1'})$

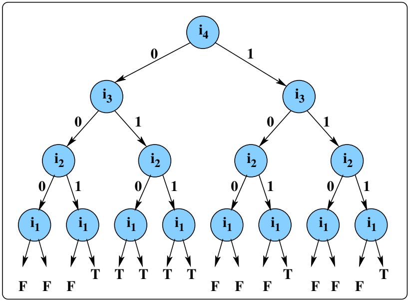
P_2 is $(i_2 \wedge w_2' \wedge \overline{i_2'}) \vee (w_2 \wedge i_1 \wedge a_2' \wedge \overline{w_2'}) \vee (w_2 \wedge w_1 \wedge t \wedge a_2' \wedge \overline{w_2'}) \vee (a_2 \wedge i_2' \wedge \overline{a_2'})$

P_3 is $(i_1 \wedge w_2 \wedge t') \vee (i_2 \wedge w_1 \wedge \overline{t'})$

Efficiently encoding transition relation

Encode as an ordered binary decision tree (OBDT):

- The levels denote the different variables, and paths through the tree represent valuations of the transition relation. The OBDT for $(i_1 \wedge i_2) \vee (i_3 \wedge \overline{i_4})$:



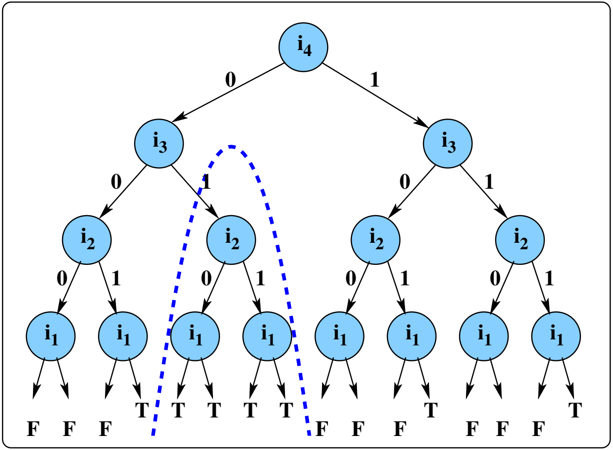
Efficiently encoding transition relation

From OBDT to ROBDD:

- Note that if we reorder the variables, we get a different decision tree, but this new tree still represents the predicate.
- In other words, it is independent of the order of the variables.
- The OBDT does not scale well, but there are optimizations that may be done.
- An optimization to exploit repetition on OBDTs leads to reduced ordered binary decision diagrams (ROBDDs).

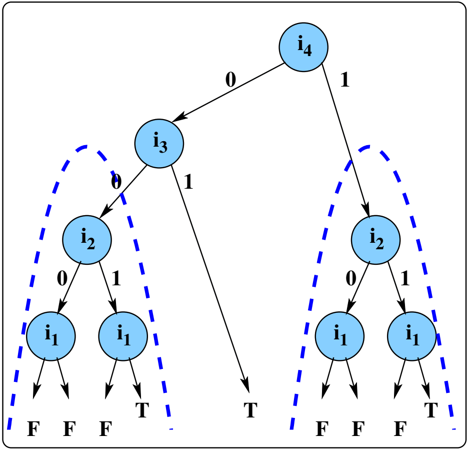
ROBDD reduction from OBDT

Remove ineffective subtree:



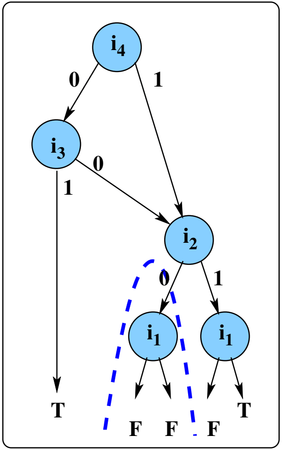
ROBDD reduction from OBDT

Identify and merge duplicate subtrees:



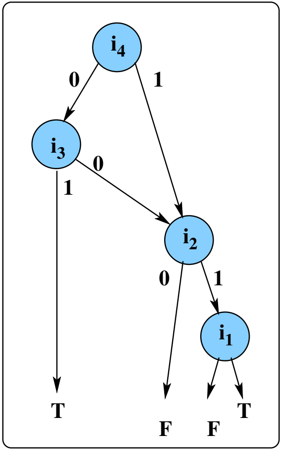
ROBDD reduction from OBDT

Remove ineffective subtree:



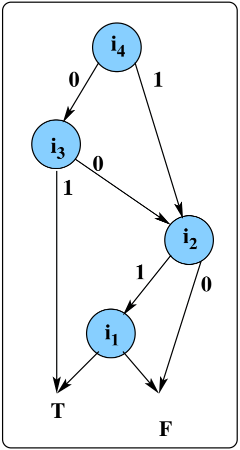
ROBDD reduction from OBDT

Merge common paths:



ROBDD reduction from OBDT

Final reduced tree:



ROBDD reduction from OBDT

A significant optimization:

ROBDDs provide a canonical form for the OBDTs, but more significantly, similar sub-trees of a OBDT result in the ROBDD merging the two subtrees.

Bryant introduced these data structures, showing how such representations of functions may be manipulated efficiently. In the paper, fast algorithms for common boolean operations are described, with complexities proportional to the sizes of the graphs.

The ROBDD optimization for the purpose of model checking was first identified by McMillan, and resulted in significant improvements in the number of states that could be model-checked.