

Verification of Real Time Systems - CS5270

9th lecture

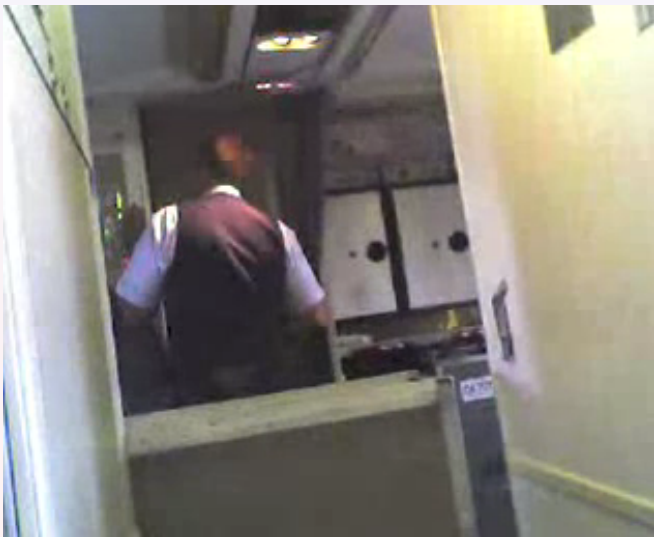
Hugh Anderson

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



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



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



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! ...

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



The immediate road map

The topics:

- **TTS: Timed transition systems**
 - Reduction: $TTS \rightarrow TS_{TTS} \rightarrow TA_{TTS} \rightarrow QTS$, regions and zones (zone operations, DBMs)
- **Preliminaries for Model Checking**
 - Behaviour, safety, liveness, automata, reachability
- **Model Checking**
 - Temporal logic
 - Foundations for CTL/TCTL model checking (Kripke semantics)
- **Model Checking**
 - The model checking relation
 - The model checking algorithm, with optimizations

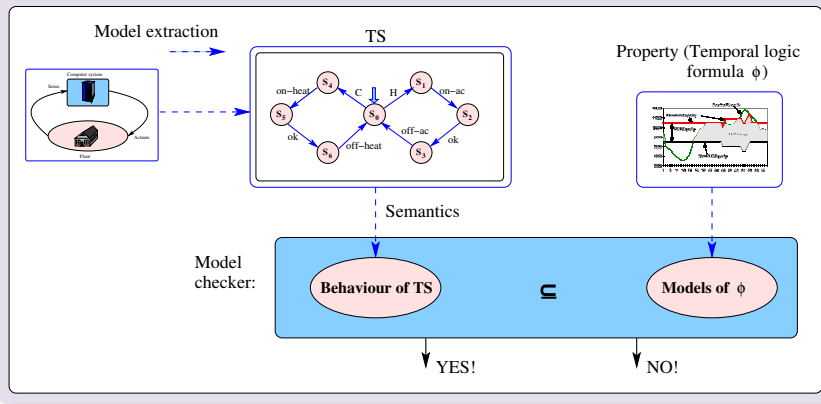
Outline

- 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



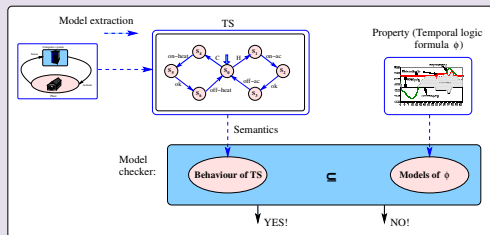
The big picture...

Properties and behaviour:



The big picture...

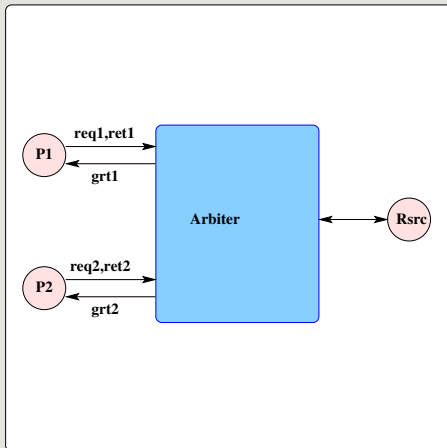
Properties and behaviour:



- **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**.

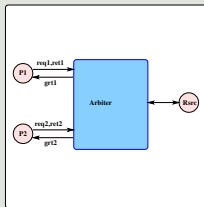
A simple system

Resource arbiter:



A simple system

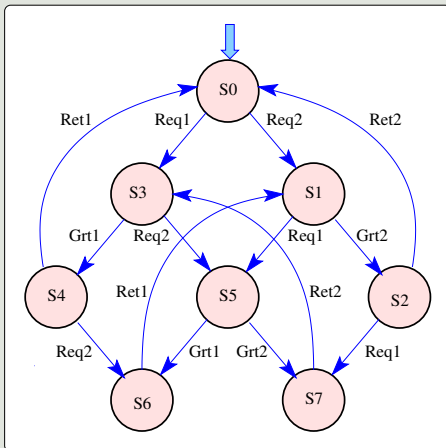
Resource arbiter:



- **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.

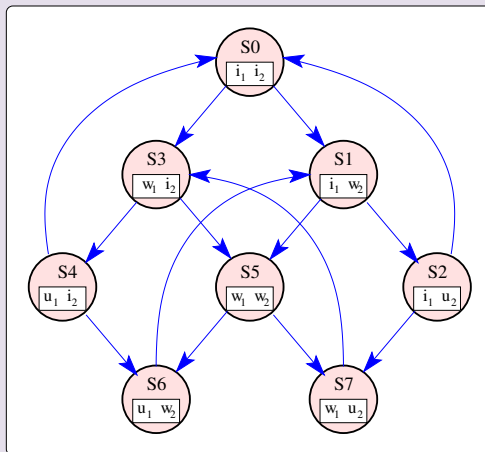
Outline

- 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



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:

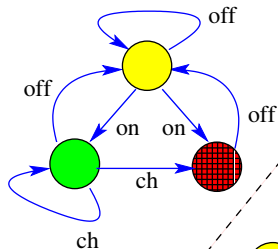
- we have $AP = \{i_1, w_1, u_1, i_2, w_2, u_2\}$
- Write out $\mathcal{L}(s)$ for each state s . The labelling function $\mathcal{L} : S \rightarrow 2^{AP}$:

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

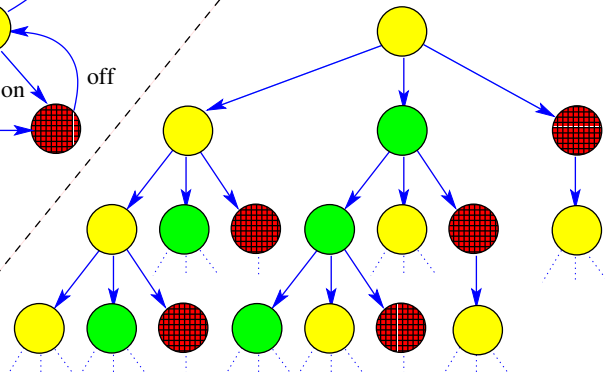
Unfolding the Kripke structure

Easier to visualize $UF(K)$:

TS (K if you ignore the actions)

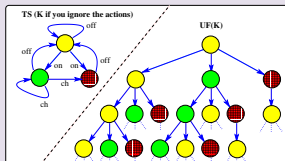


$UF(K)$



Unfolding the Kripke structure

Definition:



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

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

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

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



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:
 - $EX p$: For one computation path, property p holds in the **next** state;
 - $A(pU q)$: For all computation paths, property p holds until q holds.
 - $E(pU q)$: For one computation path, property p holds until q holds.
- (Call this **CTL-**)

CTL- formulae

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- formulae, 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
- $EX(\psi_1)$ is a CTL- formula
- $A(\psi_1 U \psi_2)$ is a CTL- formula
- $E(\psi_1 U \psi_2)$ is a CTL- formula

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 formulae
 - Semantics of CTL - the modelling relation
 - The model checking algorithm



Semantics of CTL-

Expressed in terms of model and modelling relation...

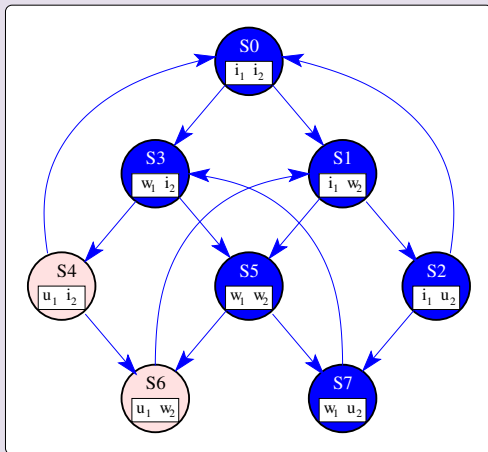
- Model checking is commonly expressed as a ternary relation (\models):

$$M, s \models P$$

- 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 $EX(w_1)$...

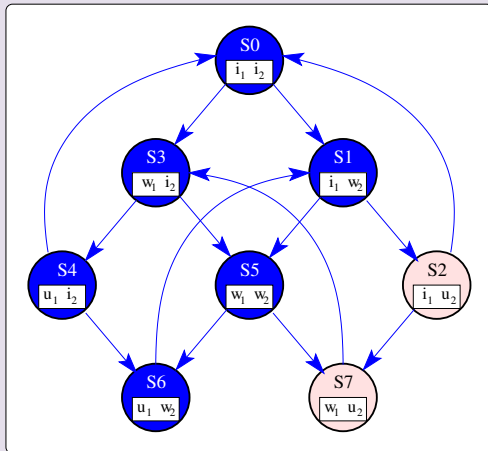
States coloured blue have desired temporal formula...



$M, s_0 \models EX(w_1)?$

Labelling the system for $E(i_2 U w_2) \dots$

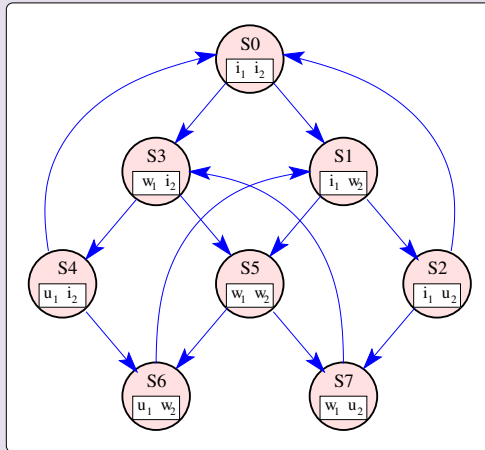
States coloured blue have desired temporal formula...



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

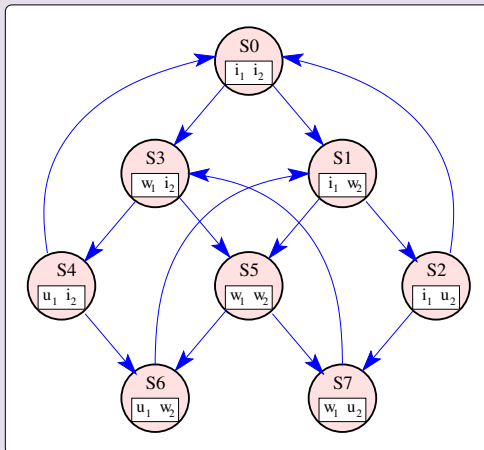
$$M, s_2 \models E(u_2 \cup w_1)?$$

Label states, check inclusion...



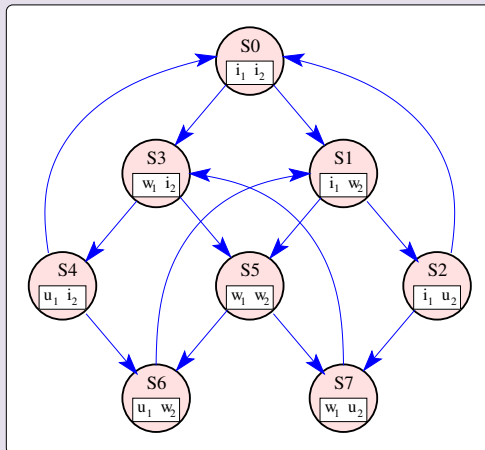
$$M, s_2 \models A(u_2 \cup w_1)?$$

Label states, check inclusion...



$$M, s_2 \models A(u_2 \cup i_2)?$$

Label states, check inclusion...



Inductive definition of the modelling relation

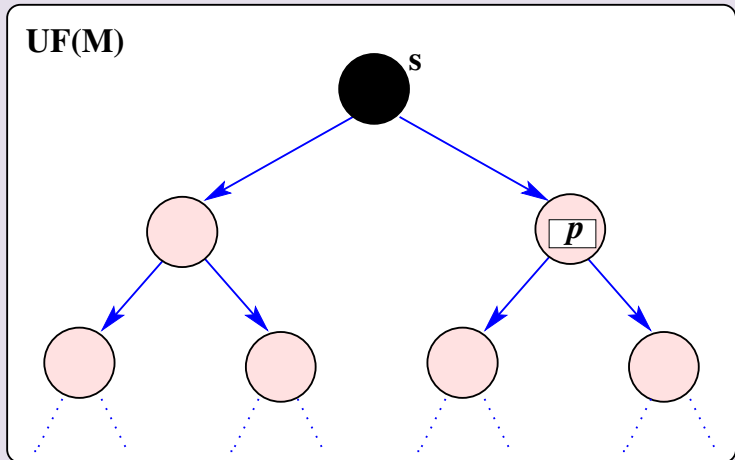
The model checking relation is defined for...

...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 $M, s \models \psi_2$
$M, s \models \text{EX}(\psi_1)$	\Leftrightarrow	iff $\Delta(s, s')$ and $M, s' \models \psi_1$ (i.e. s has a successor state at which ψ_1 holds)
$M, s \models \text{A}(\psi_1 \text{ U } \psi_2)$	\Leftrightarrow	iff for every path $\pi = s_0 s_1 \dots$ from s , for some j , $M, \pi(j) \models \psi_2$, and $\forall i < j$ $M, \pi(i) \models \psi_1$
$M, s \models \text{E}(\psi_1 \text{ U } \psi_2)$	\Leftrightarrow	iff there is a path $\pi = s_0 s_1 \dots$ from s , where for some j , $M, \pi(j) \models \psi_2$, and $\forall i < j$ $M, \pi(i) \models \psi_1$

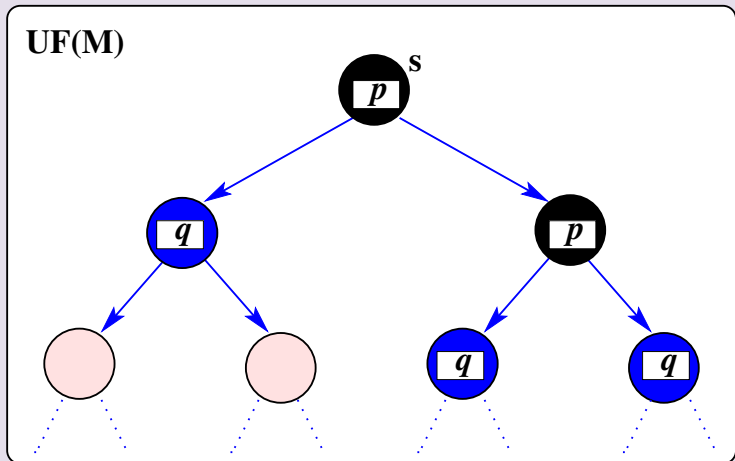
Temporal CTL operator $M, s \models \text{EX}(p)$ in $\text{UF}(K)$

Easier to see when unfolded:



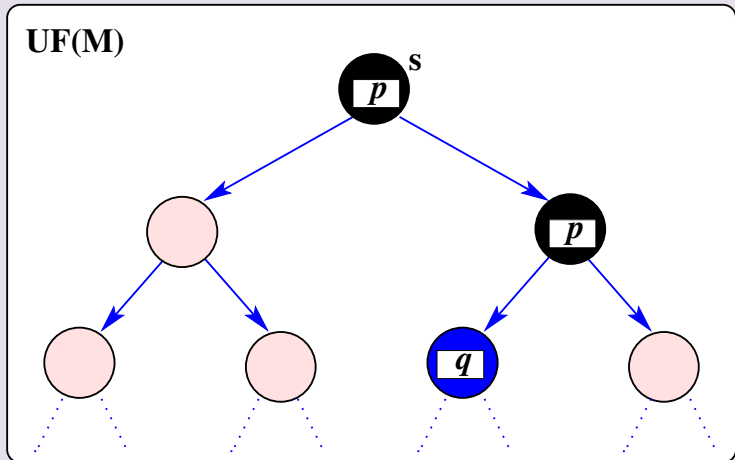
Temporal CTL operator $M, s \models A(pUq)$ in $UF(K)$

Easier to see when unfolded:



Temporal CTL operator $M, s \models E(p \cup q)$ in $UF(K)$

Easier to see when unfolded:



Defining CTL operators in CTL-

Two of the missing operators:

- $AX(\psi) = \neg EX(\neg\psi)$ For every next state ψ holds. It is not the case that there exists a next state at which ψ does not hold.
- $EG(\psi) = \neg A(\text{true} \cup \neg\psi)$ 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 ...

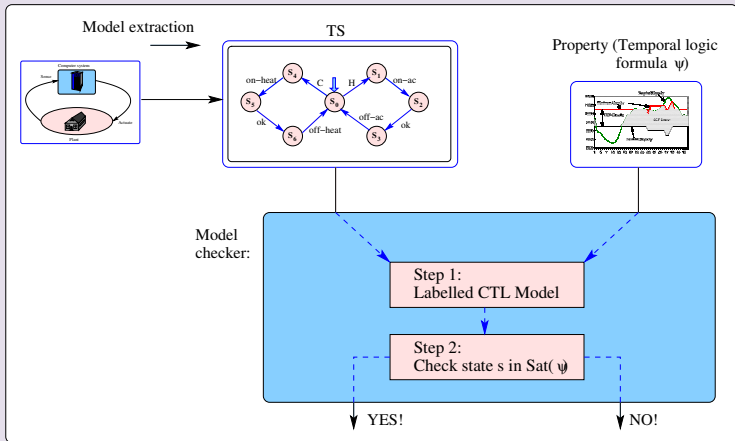
Outline

- 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



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  $\psi$ ) =
  if  $\psi \in AP$  then   $\{s \mid \psi \in \mathcal{L}(s)\}$ 
  else case  $\psi$  of
    true:           S
    false:           $\emptyset$ 
     $\neg\psi$ :           $S - \text{sat}(\psi)$ 
     $\psi_1 \wedge \psi_2$ :    $\text{sat}(\psi_1) \cap \text{sat}(\psi_2)$ 
     $\psi_1 \vee \psi_2$ :    $\text{sat}(\psi_1) \cup \text{sat}(\psi_2)$ 
     $\text{EX}(\psi_1)$ :        $\{s \in S \mid s' \in s^\uparrow \wedge s' \in \text{sat}(\psi_1)\}$ 
     $\text{A}(\psi_1 \text{ U } \psi_2)$ :  $\text{lfp}(g(Z) = \text{sat}(\psi_2) \cup (\text{sat}(\psi_1) \cap \{s \in S \mid \forall s' \in s^\uparrow \cap Z\}))$ 
     $\text{E}(\psi_1 \text{ U } \psi_2)$ :  $\text{lfp}(h(Z) = \text{sat}(\psi_2) \cup (\text{sat}(\psi_1) \cap \{s \in S \mid \exists s' \in s^\uparrow \cap Z\}))$ 
  
```


The satisfaction function for CTL model checking

Least fix-point:

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

$$A(\psi_1 U \psi_2) = \psi_2 \vee (\psi_1 \wedge AX(A(\psi_1 U \psi_2))), \text{ and}$$

$$E(\psi_1 U \psi_2) = \psi_2 \vee (\psi_1 \wedge EX(E(\psi_1 U \psi_2)))$$

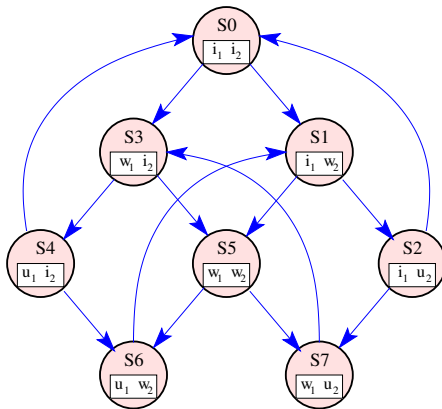
Express as fix-points of the corresponding functions

$$g(Z) = \psi_2 \vee (\psi_1 \wedge AX(Z)), \text{ and}$$

$$h(Z) = \psi_2 \vee (\psi_1 \wedge EX(Z))$$

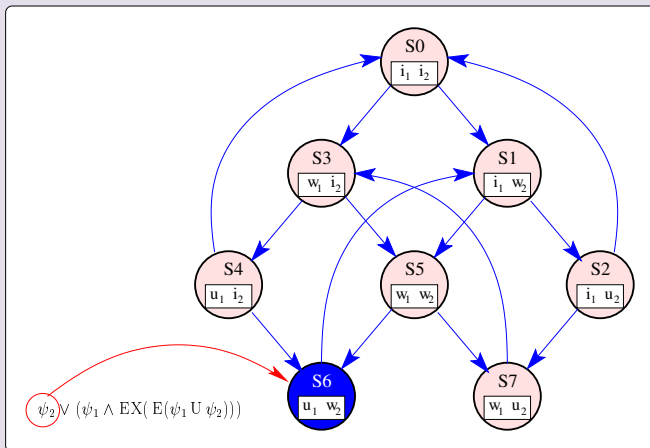
Checking $M, s_2 \models E(i_2 U (u_1 \wedge w_2)) \dots$

Start with the labelled Kripke structure:



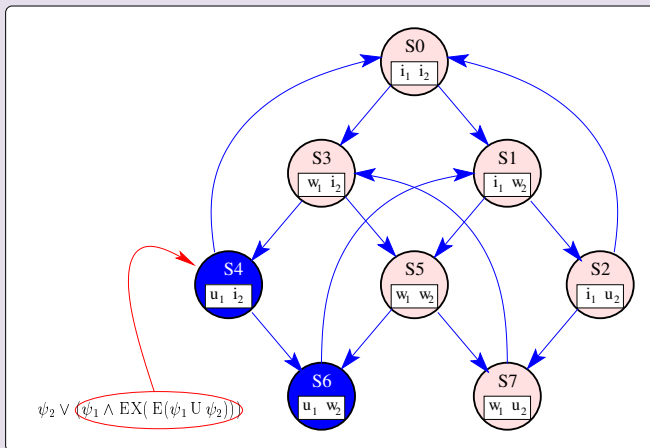
Checking $\text{sat}(E(i_2 \cup (u_1 \wedge w_2))) \dots$

Using lfp equation:



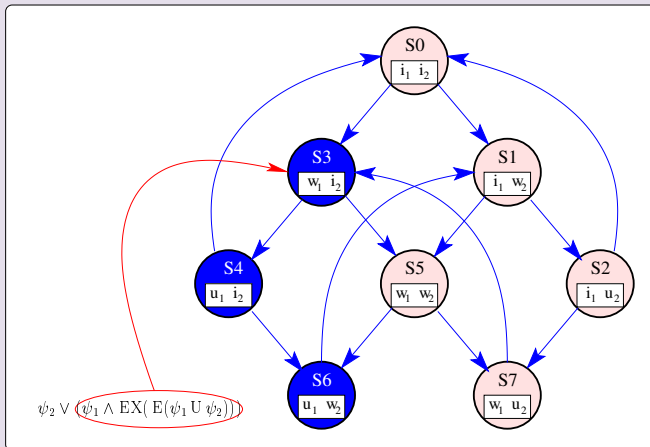
Checking $\text{sat}(E(i_2 \text{ U } (u_1 \wedge w_2))) \dots$

Using lfp equation:



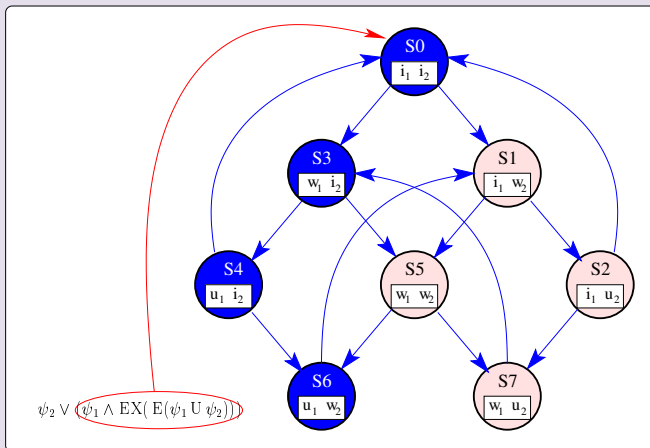
Checking $\text{sat}(E(i_2 \text{ U } (u_1 \wedge w_2))) \dots$

Using lfp equation:



Checking $\text{sat}(E(i_2 \text{ U } (u_1 \wedge w_2))) \dots$

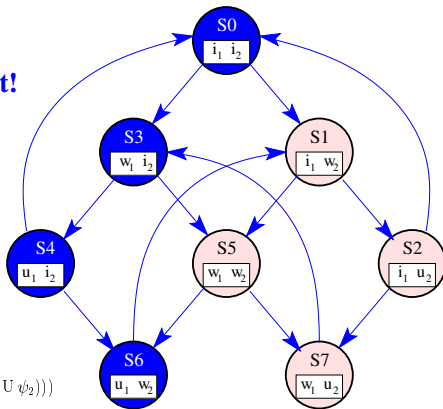
Using lfp equation:



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

Once we reach the fix-point:

Fixed point!



$$\psi_2 \vee (\psi_1 \wedge \text{EX}(\text{E}(\psi_1 U \psi_2)))$$

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**.

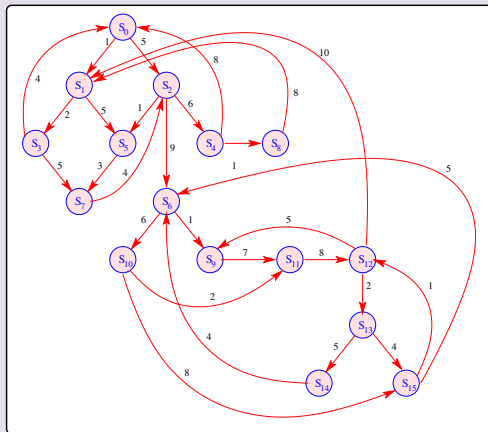
A *different* arbiter

The source code:

```
P1  = if idle1 then (wait1 := 1; idle1 := 0) else  
      if wait1 ∧ idle2 then (active1 := 1; wait1 := 0) else  
      if wait1 ∧ wait2 ∧ ¬turn then (active1 := 1; wait1 := 0);  
      if active1 then (CritSect; idle1 := 1; active1 := 0);  
P2  = if idle2 then (wait2 := 1; idle2 := 0) else  
      if wait2 ∧ idle1 then (active2 := 1; wait2 := 0) else  
      if wait2 ∧ wait1 ∧ turn then (active2 := 1; wait2 := 0);  
      if active2 then (CritSect; idle2 := 1; active2 := 0);  
T1  = if idle1 ∧ wait2 then turn := 1 else  
      if idle2 ∧ wait1 then turn := 0;  
System = (P1 || P2 || T1); System;
```

Transition diagram

Numbers relate back to program:



How do we get this?

Encoding states as boolean formulae:

- 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 :
 - $S = \{000, 001, 010, 011, 100, 101, 110, 111\}$
 - $S =$
 $\{\neg a \wedge \neg b \wedge \neg c, \neg a \wedge \neg b \wedge c, \neg a \wedge b \wedge \neg c, \neg a \wedge b \wedge c, \dots, a \wedge b \wedge c\}$
- 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 $(i_1 \wedge w'_1 \wedge \bar{i}'_1) \vee (w_1 \wedge i_2 \wedge a'_1 \wedge \bar{w}'_1) \vee (w_1 \wedge w_2 \wedge \bar{t} \wedge a'_1 \wedge \bar{w}'_1) \vee (a_1 \wedge i'_1 \wedge \bar{a}'_1)$

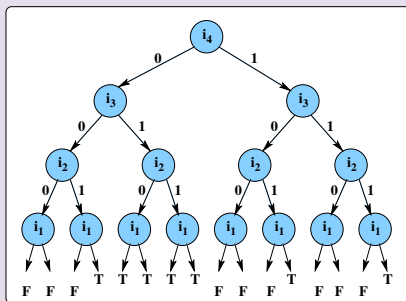
P_2 is $(i_2 \wedge w'_2 \wedge \bar{i}'_2) \vee (w_2 \wedge i_1 \wedge a'_2 \wedge \bar{w}'_2) \vee (w_2 \wedge w_1 \wedge t \wedge a'_2 \wedge \bar{w}'_2) \vee (a_2 \wedge i'_2 \wedge \bar{a}'_2)$

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

Efficiently encoding transition relation

Encode as an ordered binary decision tree (OBDD):

- The levels denote the different variables, and paths through the tree represent valuations of the transition relation. The OBDD for $(i_1 \wedge i_2) \vee (i_3 \wedge \bar{i}_4)$:



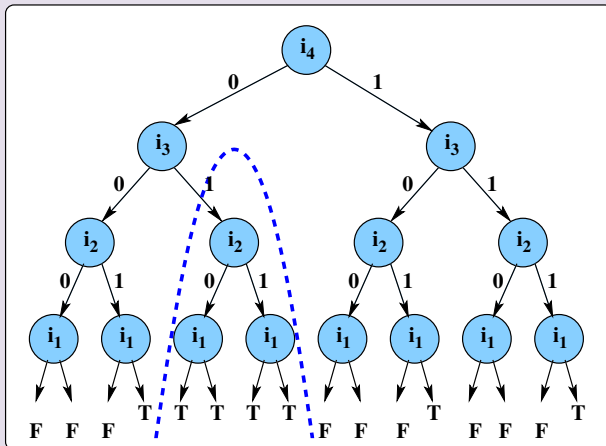
Efficiently encoding transition relation

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).

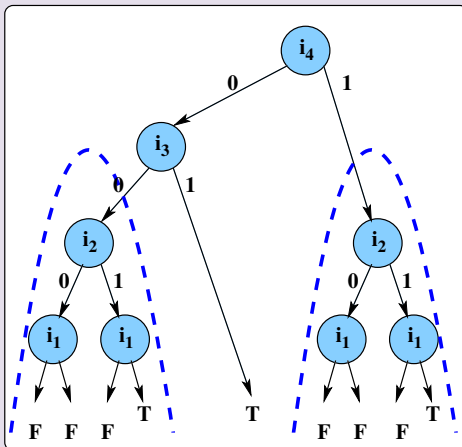
ROBDD reduction from OBDT

Remove ineffective subtree:



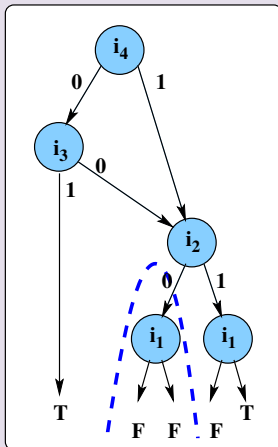
ROBDD reduction from OBDT

Identify and merge duplicate subtrees:



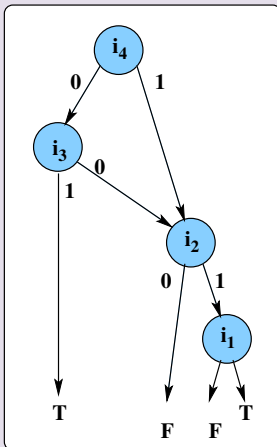
ROBDD reduction from OBDT

Remove ineffective subtree:



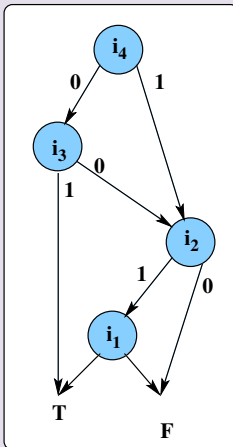
ROBDD reduction from OBDT

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.