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

Hugh Anderson  Verification of Real Time Systems - CS5270 9th lecture  3
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: $\text{TTS} \rightarrow \text{TS}_{\text{TTS}} \rightarrow \text{TA}_{\text{TTS}} \rightarrow \text{QTS}$, 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
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:

Model extraction

TS

Behaviour of TS

Models of \( \phi \)

Model checker:

YES!

Property (Temporal logic formula \( \phi \))

Model extraction

TS

Behaviour of TS

Models of \( \phi \)

Model checker:

YES!

Model extraction

TS

Behaviour of TS

Models of \( \phi \)

Model checker:

YES!

Hugh Anderson

Verification of Real Time Systems - CS5270 9th lecture
**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 $\phi$, returning **YES** or **NO**.
A simple system

Resource arbiter:

```
P1
req1,ret1
grt1

Rsrc

P2
req2,ret2
grt2

Arbiter

Hugh Anderson
Verification of Real Time Systems - CS5270 9th lecture
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:

```
S0
S1
S2
S3
S4 S5
S6 S7

Req1
Req2
Ret2
Ret1

Grt1
Grt2

S0 -> S1
S1 -> S2
S2 -> S3
S3 -> S4
S4 -> S5
S5 -> S6
S6 -> S7
S7 -> S0
```

Hugh Anderson
Verification of Real Time Systems - CS5270 9th lecture
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...

The Kripke structure...
A formal semantics for modal logic systems:

The □ 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 $(S, \Delta, AP, L)$, 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
- $L : S \rightarrow 2^{AP}$ 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 $L(s)$ for each state $s$. The labelling function $L : S \rightarrow 2^{AP}$:

$$L = \{(s_0, \{i_1, i_2\}),
(s_1, \{i_1, w_2\}),
(s_2, \{i_1, u_2\}),
(s_3, \{w_1, i_2\}),
(s_4, \{u_1, i_2\}),
(s_5, \{w_1, w_2\}),
(s_6, \{u_1, w_2\}),
(s_7, \{w_1, u_2\})\}$$
Unfolding the Kripke structure

Easier to visualize UF(K):

TS (K if you ignore the actions)

off

off

off

on

on

off

ch

ch

UF(K)
Unfolding the Kripke structure

**Definition:**

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

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

$\text{UF}(\mathcal{K})$ is another Kripke structure.
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 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;
  - $A(p U q)$: For all computation paths, property $p$ holds until $q$ holds.
  - $E(p U q)$: 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 \( \psi_1 \) and \( \psi_2 \) are CTL- formulæ, then

- \( \neg \psi_1 \) is a CTL- formula
- \( \psi_1 \land \psi_2 \) is a CTL- formula
- \( \psi_1 \lor \psi_2 \) is a CTL- formula
- \( \text{EX}(\psi_1) \) is a CTL- formula
- \( \text{A}(\psi_1 \cup \psi_2) \) is a CTL- formula
- \( \text{E}(\psi_1 \cup \psi_2) \) is a CTL- formula
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 \textit{inductively}, with a set of interlocking rules.

- A \textit{labelling} algorithm may then be used to establish the set of states satisfying the relation.
Labelling the system for $\text{EX}(w_1)$...

States coloured blue have desired temporal formula...

$M, s_0 \models \text{EX}(w_1)$?
Labelling the system for $E(i_2 U w_2)$...

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...

```plaintext
S0
S1
S2
S3
S4
S5
S6
S7

$\begin{align*}
M & = E(u_2 \cup w_1) \\
S_0 & = \{i_1, i_2\} \\
S_1 & = \{i_1, w_2\} \\
S_2 & = \{i_1, u_2\} \\
S_3 & = \{w_1, i_2\} \\
S_4 & = \{u_1, i_2\} \\
S_5 & = \{w_1, w_2\} \\
S_6 & = \{u_1, w_2\} \\
S_7 & = \{w_1, u_2\}
\end{align*}$
```
$M, s_2 \models A(u_2 U w_1)$?

Label states, check inclusion...

Diagram with states S0, S1, S2, S3, S4, S5, S6, S7, and edges connecting them with labels $u_1, u_2, w_1, w_2$.
$M, s_2 \models A(u_2 U i_2)$?

Label states, check inclusion...

Diagram showing states $S0$ to $S7$ with labels $i_1 i_2$, $w_1 i_2$, $u_1 i_2$, $w_1 w_2$, $i_1 w_2$, $u_1 w_2$, $w_1 u_2$, $i_1 u_2$.
### Inductive definition of the modelling relation

The model checking relation is defined for...

...each atomic proposition $p$ and each CTL-formula $\psi_1, \psi_2$ as:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M, s \models p$</td>
<td>$\iff p \in L(s)$</td>
</tr>
<tr>
<td>$M, s \models \neg \psi_1$</td>
<td>$\iff$ it is not the case that $M, s \models \psi_1$</td>
</tr>
<tr>
<td>$M, s \models \psi_1 \land \psi_2$</td>
<td>$\iff$ if $M, s \models \psi_1$ and $M, s \models \psi_2$</td>
</tr>
<tr>
<td>$M, s \models \psi_1 \lor \psi_2$</td>
<td>$\iff$ if $M, s \models \psi_1$ or $M, s \models \psi_2$</td>
</tr>
<tr>
<td>$M, s \models \text{EX}(\psi_1)$</td>
<td>$\iff$ if $\Delta(s, s')$ and $M, s' \models \psi_1$</td>
</tr>
<tr>
<td></td>
<td>(i.e. $s$ has a successor state at which $\psi_1$ holds)</td>
</tr>
<tr>
<td>$M, s \models \text{A}(\psi_1 \lor \psi_2)$</td>
<td>$\iff$ if for every path $\pi = s_0 , s_1 \ldots$ from $s$, for</td>
</tr>
<tr>
<td></td>
<td>some $j$, $M, \pi(j) \models \psi_2$, and $\forall i &lt; j , M, \pi(i) \models \psi_1$</td>
</tr>
<tr>
<td>$M, s \models \text{E}(\psi_1 \lor \psi_2)$</td>
<td>$\iff$ if there is a path $\pi = s_0 , s_1 \ldots$ from $s$, where for</td>
</tr>
<tr>
<td></td>
<td>some $j$, $M, \pi(j) \models \psi_2$, and $\forall i &lt; j , M, \pi(i) \models \psi_1$</td>
</tr>
</tbody>
</table>
Temporal CTL operator $M, s \models EX(p)$ in $UF(K)$

Easier to see when unfolded:
Temporal CTL operator $M, s \models A(p U q)$ in UF(K)

Easier to see when unfolded:
Temporal CTL operator $M, s \models E(p U q)$ in UF(K)

Easier to see when unfolded:
Two of the missing operators:

- $AX(\psi) = \neg EX(\neg \psi)$ For every next state $\psi$ holds. It is not the case that there exists a next state at which $\psi$ does not hold.

- $EG(\psi) = \neg A(true U \neg \psi)$ There exists a path $\pi$ 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:

Model extraction → TS → Property (Temporal logic formula $\psi$)

Model checker:

- Step 1: Labelled CTL Model
- Step 2: Check state $s$ in $\text{Sat}(\psi)$

YES! → NO!
The satisfaction function for CTL model checking

Returns a set of states:

\[
\text{set_of_States sat(Property } \psi) =
\begin{align*}
\text{if } \psi \in AP & \text{ then } \{s \mid \psi \in \mathcal{L}(s)\} \\
\text{else case } \psi \text{ of} & \\
\text{true: } & S \\
\text{false: } & \emptyset \\
\neg \psi: & S - \text{sat(}\psi) \\
\psi_1 \land \psi_2: & \text{sat(}\psi_1) \cap \text{sat(}\psi_2) \\
\psi_1 \lor \psi_2: & \text{sat(}\psi_1) \cup \text{sat(}\psi_2) \\
\text{EX}(\psi_1): & \{s \in S \mid s' \in s^\uparrow \land s' \in \text{sat(}\psi_1)\} \\
\text{A}(\psi_1 \cup \psi_2): & \text{lfp}(g(Z) = \text{sat(}\psi_2) \cup (\text{sat(} \psi_1) \cap \{s \in S \mid \forall s' \in s^\uparrow \cap Z\})) \\
\text{E}(\psi_1 \cup \psi_2): & \text{lfp}(h(Z) = \text{sat(}\psi_2) \cup (\text{sat(} \psi_1) \cap \{s \in S \mid \exists s' \in s^\uparrow \cap Z\}))
\end{align*}
\]
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 $\text{sat}_{AU}$ and $\text{sat}_{EU}$). What are the functions $g$ and $h$? Some investigation will show that

$$
A(\psi_1 U \psi_2) = \psi_2 \lor (\psi_1 \land AX(A(\psi_1 U \psi_2))), \text{ and}
$$

$$
E(\psi_1 U \psi_2) = \psi_2 \lor (\psi_1 \land EX(E(\psi_1 U \psi_2)))
$$

Express as fix-points of the corresponding functions

$$
g(Z) = \psi_2 \lor (\psi_1 \land AX(Z)), \text{ and}
$$

$$
h(Z) = \psi_2 \lor (\psi_1 \land EX(Z))
$$
Checking $M, s_2 \models E(i_2 U (u_1 \land w_2))$...

Start with the labelled Kripke structure:

- $S_0$: $i_1, i_2$
- $S_1$: $i_1, w_2$
- $S_2$: $i_1, u_2$
- $S_3$: $w_1, i_2$
- $S_4$: $u_1, i_2$
- $S_5$: $w_1, w_2$
- $S_6$: $u_1, w_2$
- $S_7$: $w_1, u_2$
Using lfp equation:

Checking $\text{sat}(E(i_2 U (u_1 \land w_2)))$...

$$\psi_2 \lor (\psi_1 \land \text{EX}(E(\psi_1 U \psi_2)))$$
Checking $\text{sat}(E(i_2 U (u_1 \land w_2)))...$

Using lfp equation:

$\psi_2 \lor (\psi_1 \land \text{EX}(E(\psi_1 U \psi_2)))$
Using lfp equation:

\[ \psi_2 \lor (\psi_1 \land \text{EX} (E(\psi_1 \lor \psi_2))) \]
Checking $\text{sat}(E(i_2 U (u_1 \land w_2)))$...

Using lfp equation:

\[
\psi_2 \lor (\psi_1 \land \text{EX}(E(\psi_1 U \psi_2)))
\]
Once we reach the fix-point:

\[
M, s_2 \models E(i_2 U (u_1 \land w_2)) \quad \text{if} \quad s_2 \in \text{sat}(E(i_2 U (u_1 \land w_2)))
\]

\[
\psi_2 \lor (\psi_1 \land \text{EX}(E(\psi_1 U \psi_2)))
\]
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:

\[
P_1 = \begin{cases} \text{if } \text{idle}_1 \text{ then } (\text{wait}_1 := 1; \text{idle}_1 := 0) & \text{else} \\ \text{if } \text{wait}_1 \land \text{idle}_2 \text{ then } (\text{active}_1 := 1; \text{wait}_1 := 0) & \text{else} \\ \text{if } \text{wait}_1 \land \text{wait}_2 \land \neg \text{turn} \text{ then } (\text{active}_1 := 1; \text{wait}_1 := 0) & \text{else} \\ \text{if } \text{active}_1 \text{ then } (\text{CritSect}; \text{idle}_1 := 1; \text{active}_1 := 0) & \end{cases}
\]

\[
P_2 = \begin{cases} \text{if } \text{idle}_2 \text{ then } (\text{wait}_2 := 1; \text{idle}_2 := 0) & \text{else} \\ \text{if } \text{wait}_2 \land \text{idle}_1 \text{ then } (\text{active}_2 := 1; \text{wait}_2 := 0) & \text{else} \\ \text{if } \text{wait}_2 \land \text{wait}_1 \land \text{turn} \text{ then } (\text{active}_2 := 1; \text{wait}_2 := 0) & \text{else} \\ \text{if } \text{active}_2 \text{ then } (\text{CritSect}; \text{idle}_2 := 1; \text{active}_2 := 0) & \end{cases}
\]

\[
T_1 = \begin{cases} \text{if } \text{idle}_1 \land \text{wait}_2 \text{ then } \text{turn} := 1 & \text{else} \\ \text{if } \text{idle}_2 \land \text{wait}_1 \text{ then } \text{turn} := 0 & \end{cases}
\]

System = (P_1 \parallel P_2 \parallel T_1); \text{ System;}

Hugh Anderson
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 $a, b, c$:
  - $S = \{000, 001, 010, 011, 100, 101, 110, 111\}$
  - $S = \{\neg a \land \neg b \land \neg c, \neg a \land \neg b \land c, \neg a \land b \land \neg c, \neg a \land b \land c, \ldots, a \land b \land c\}$

- Encode transitions using before $(a, b, c)$ and after $(a', b', c')$ variables.
  - For example: $(s_1, s_4) = (\neg a \land \neg b \land \neg c) \land (\neg a' \land b' \land c')$
Transition relation as a predicate

Transition system ends up as a boolean formula:

- $P_1$ is $(i_1 \land w'_1 \land i'_2) \lor (w_1 \land i_2 \land a'_1 \land \overline{w'_1}) \lor (w_1 \land w_2 \land \overline{t} \land a'_1 \land \overline{w'_1}) \lor (a_1 \land i'_1 \land \overline{a'_1})$

- $P_2$ is $(i_2 \land w'_2 \land i'_2) \lor (w_2 \land i_1 \land a'_2 \land \overline{w'_2}) \lor (w_2 \land w_1 \land t \land a'_2 \land \overline{w'_2}) \lor (a_2 \land i'_2 \land \overline{a'_2})$

- $P_3$ is $(i_1 \land w_2 \land t') \lor (i_2 \land w_1 \land \overline{t'})$
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 \((i_1 \land i_2) \lor (i_3 \land i_4)\):

```
  i4
   /\  \\
  i3   i3
   /\  /\  \\
 i2   i2   i2   i2
    /\  /\  /\  /\  \\
 i1   i1   i1   i1   i1
```

```
F F F T T T T
F F F T T T T
F F F T T T T
T T T T T T T
T T T T T T T
T T T T T T T
T T T T T T T
T T T T T T T
```
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:

```
F F F T T T T T
```

Hugh Anderson
Verification of Real Time Systems - CS5270 9th lecture
ROBDD reduction from OBDT

Identify and merge duplicate subtrees:
Remove ineffective subtree:

![Diagram of ROBDD reduction from OBDT]

- The diagram shows a ROBDD (Read-Only Binary Decision Diagram) with nodes labeled as follows:
  - i1
  - i2
  - i3
  - i4

- The ROBDD is reduced by removing ineffective subtrees based on the truth values of the boolean function:
  - T
  - F

- The reduction process involves removing subtrees that are not relevant to the final truth value of the function.
Merge common paths:

```
- i1
- i2
- i3
- i4

Transition:
- T 0 1 0 1
- F
```

Hugh Anderson
Verification of Real Time Systems - CS5270 9th lecture 49
Final reduced tree:
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.