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:

  ![Diagram of a computation 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 \(\pi : \mathbb{N} \rightarrow S\):

- \(\pi(i)\) is the \(i\)th element of \(\pi\) (first element is \(\pi(0)\))
- might sometimes write \(\pi \downarrow i\) instead of \(\pi(i)\)
- \(\pi \downarrow i\) is the \(i\)-th tail of \(\pi\) 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 \(\pi\) is a path starting at \(s\):

Path \[ R_s \pi = (\pi(0) = s) \land \forall i. \ R(\pi(i)) (\pi(i+1)) \]

where:

Path : \((S \rightarrow S \rightarrow \mathbb{B}) \rightarrow S \rightarrow (\mathbb{N} \rightarrow S) \rightarrow \mathbb{B}\)

transition relation
initial state
path
Consider this timing diagram:

- Two handshake properties representing the diagram:
  - following a rising edge on \( \text{dreq} \), the value of \( \text{dreq} \) remains 1 (i.e. \text{true}) until it is acknowledged by a rising edge on \( \text{dack} \)
  - following a falling edge on \( \text{dreq} \), the value on \( \text{dreq} \) remains 0 (i.e. \text{false}) until the value of \( \text{dack} \) is 0

A property language is used to formalise such properties.
DIV: example program properties

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 $\textbf{JM1}$: a non-deterministic program example

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: IF LOCK=0 THEN LOCK:=1;</td>
<td>0: IF LOCK=0 THEN LOCK:=1;</td>
</tr>
<tr>
<td>1: X:=1;</td>
<td>1: X:=2;</td>
</tr>
<tr>
<td>2: IF LOCK=1 THEN LOCK:=0;</td>
<td>2: IF LOCK=1 THEN LOCK:=0;</td>
</tr>
<tr>
<td>3:</td>
<td>3:</td>
</tr>
</tbody>
</table>

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

$\forall pc_1 \ pc_2 \ lock \ x. R_{\text{JM1}} (0, pc_2, 0, x) \ (1, pc_2, 1, x) \land$
$R_{\text{JM1}} (1, pc_2, lock, x) (2, pc_2, lock, 1) \land$
$R_{\text{JM1}} (2, pc_2, 1, x) \ (3, pc_2, 0, x) \land$
$R_{\text{JM1}} (pc_1, 0, 0, x) \ (pc_1, 1, 1, x) \land$
$R_{\text{JM1}} (pc_1, 1, lock, x) (pc_1, 2, lock, 2) \land$
$R_{\text{JM1}} (pc_1, 2, 1, x) \ (pc_1, 3, 0, x)$

- An atomic property:
  - $\text{NotAt11}(pc_1, pc_2, lock, x) = \neg((pc_1 = 1) \land (pc_2 = 1))$

- A non-atomic property:
  - all states reachable from $(0, 0, 0, 0)$ satisfy $\text{NotAt11}$
  - this is an example of a reachability property
State satisfying \textbf{NotAt11} unreachable from \((0,0,0,0)\)

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: IF LOCK=0 THEN LOCK:=1;</td>
<td>0: IF LOCK=0 THEN LOCK:=1;</td>
</tr>
<tr>
<td>1: X:=1;</td>
<td>1: X:=2;</td>
</tr>
<tr>
<td>2: IF LOCK=1 THEN LOCK:=0;</td>
<td>2: IF LOCK=1 THEN LOCK:=0;</td>
</tr>
<tr>
<td>3:</td>
<td></td>
</tr>
</tbody>
</table>

\[ R_{JM1} (0, pc_2, 0, x) \begin{array}{l} (1, pc_2, 1, x) \\
R_{JM1} (1, pc_2, lock, x) \quad (2, pc_2, lock, 1) \\
R_{JM1} (2, pc_2, 1, x) \quad (3, pc_2, 0, x) \end{array} \quad R_{JM1} (pc_1, 0, 0, x) \begin{array}{l} (pc_1, 1, 1, x) \\
R_{JM1} (pc_1, 1, lock, x) \quad (pc_1, 2, lock, 2) \\
R_{JM1} (pc_1, 2, 1, x) \quad (pc_1, 3, 0, x) \end{array} \]

\[ \neg ((pc_1 = 1) \land (pc_2 = 1)) \]

- Can only reach \(pc_1 = 1 \land pc_2 = 1\) via:
  \[ R_{JM1} (0, pc_2, 0, x) \begin{array}{l} (1, pc_2, 1, x) \\
R_{JM1} (pc_1, 0, 0, x) \quad (pc_1, 1, 1, x) \end{array} \quad R_{JM1} (0, 1, 0, x) \begin{array}{l} (1, 1, 1, x) \\
R_{JM1} (0, 1, 0, x) \quad (1, 1, 1, x) \end{array} \]

- But:
  \[ R_{JM1} (pc_1, pc_2, lock, x) (pc'_1, pc'_2, lock', x') \land pc'_1=0 \land pc'_2=1 \Rightarrow lock'=1 \]
  \[ R_{JM1} (pc_1, pc_2, lock, x) (pc'_1, pc'_2, lock', x') \land pc'_1=1 \land pc'_2=0 \Rightarrow lock'=1 \]

- So can never reach \((0,1,0,x)\) or \((1,0,0,x)\)

- So can’t reach \((1,1,1,x)\), hence never \((pc_1 = 1) \land (pc_2 = 1)\)

- Hence all states reachable from \((0,0,0,0)\) satisfy \textbf{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
  
  \[
  R^0 \ s \ s' = (s = s') \\
  R^{n+1} \ s \ s' = \exists s''. \ R \ s \ s'' \land R^n \ s'' \ s'
  \]

- $R^* \ s \ s'$ means $s'$ reachable from $s$ in finite steps
  
  $R^* \ s \ s' = \exists n. \ R^n \ s \ s'$

- Note: $R^* \ s \ s' \iff \exists \pi \ n. \ Path \ R \ s \ \pi \land (s' = \pi(n))$

- The set of states reachable from $s$ is \{s' | $R^* \ s \ s'$\}

- 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 NotAt11
  
  - i.e. $\forall s'. \ R^*_{JM1} (0,0,0,0) \ s' \Rightarrow NotAt11(s')$
Models and model checking

- Assume a model \((S, R)\)
- Assume also a set \(S_0 \subseteq S\) of initial states
- Assume also a set \(AP\) of atomic properties
  - allows different models to have same atomic properties
- Assume a labelling function \(L : S \rightarrow \mathcal{P}(AP)\)
  - \(p \in L(s)\) means “\(s\) labelled with \(p\)” or “\(p\) true of \(s\)”
  - previously properties were functions \(p : S \rightarrow \mathbb{B}\)
  - now \(p \in AP\) is distinguished from \(\lambda s. \ p \in L(s)\)
  - assume \(T, F \in AP\) with forall \(s\): \(T \in L(s)\) and \(F \notin L(s)\)
- A Kripke structure is a tuple \((S, S_0, R, L)\)
  - often the term “model” is used for a Kripke structure
  - i.e. a model is \((S, S_0, R, L)\) rather than just \((S, R)\)
- Model checking computes whether \((S, S_0, R, L) \models \phi\)
  - \(\phi\) is a property expressed in a property language
  - informally \(M \models \phi\) means “wff \(\phi\) is true in model \(M\)”
Minimal property language: $\phi$ is $\text{AG}p$ where $p \in \text{AP}$

- Consider properties $\phi$ of form $\text{AG}p$ where $p \in \text{AP}$
  - “$\text{AG}$” 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 $\text{Reachable } M = \{s' \mid \exists s \in S_0. R^* s s'\}$

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

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

$$M \models \text{AG}p \iff \forall s'. s' \in \text{Reachable } M \Rightarrow p \in L(s')$$
Model checking $M \models \text{AG } p$

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

checked by:
- first computing $\text{Reachable } M$
- then checking $p$ true of all its members

- Let $S$ abbreviate $\{ s' \mid \exists s \in S_0. R^* s s' \}$ (i.e. $\text{Reachable } M$)
- Compute $S$ iteratively: $S = S_0 \cup S_1 \cup \cdots \cup S_n \cup \cdots$
  - i.e. $S = \bigcup_{n=0}^{\infty} S_n$
  - where: $S_0 = S_0$ (set of initial states)
  - and inductively: $S_{n+1} = S_n \cup \{ s' \mid \exists s \in S_n \land R s s' \}$

- Clearly $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_n \subseteq \cdots$
- Hence if $S_m = S_{m+1}$ then $S = S_m$
- Algorithm: compute $S_0, S_1, \ldots$, until no change; check all members of computed set labelled with $p$
compute $S_0, S_1, \ldots$, until no change; 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:
    $S_0 \subset S_1 \subset \cdots \subset S_n \subset \cdots$

- How to represent $S_0, S_1, \ldots$?
  - 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:

\[
M = (S_{RCV}, \{(1, 1, 1)\}, R_{RCV}, L_{RCV})
\]

and

\[
R_{RCV} (dreq, q0, dack) (dreq', q0', dack') = (q0' = dreq) \land (dack' = (dreq \land (q0 \lor dack)))
\]

AP and labelling function \(L_{RCV}\) discussed later
Possible states for $\text{RCV}$:

\{000, 001, 010, 011, 100, 101, 110, 111\}

where $b_2b_1b_0$ denotes state

$d\text{req} = b_2 \land q_0 = b_1 \land d\text{ack} = b_0$

Graph of the transition relation:
Computing \( \text{Reachable} \ M_{\text{RCV}} \)

Define:

\[
S_0 = \{ b_2 b_1 b_0 \mid b_2 b_1 b_0 \in \{111\} \} \\
= \{111\}
\]

\[
S_{i+1} = S_i \cup \{ s' \mid \exists s \in S_i. \ R_{\text{RCV}} \ s \ s' \} \\
= S_i \cup \{ b'_2 b'_1 b'_0 \mid \exists b_2 b_1 b_0 \in S_i. \ (b'_1 = b_2) \land (b'_0 = b_2 \land (b_1 \lor b_0)) \}
\]
Computing \( \text{Reachable } M_{\text{RCV}} \) (continued)

- **Compute:**

  \[ \begin{align*}
  S_0 &= \{111\} \\
  S_1 &= \{111\} \cup \{011\} \\
  &= \{111, 011\} \\
  S_2 &= \{111, 011\} \cup \{000, 100\} \\
  &= \{111, 011, 000, 100\} \\
  S_3 &= \{111, 011, 000, 100\} \cup \{010, 110\} \\
  &= \{111, 011, 000, 100, 010, 110\} \\
  S_i &= S_3 \quad (i > 3)
  \end{align*} \]

- **Hence** Reachable \( M_{\text{RCV}} = \{111, 011, 000, 100, 010, 110\} \)
Model checking $M_{RCV} \models AG\ p$

- $M = (S_{RCV}, \{111\}, R_{RCV}, L_{RCV})$

- To check $M_{RCV} \models AG\ p$
  - compute $\text{Reachable } M_{RCV} = \{111, 011, 000, 100, 010, 110\}$
  - check $\text{Reachable } M_{RCV} \subseteq \{s \mid p \in L_{RCV}(s)\}$
  - i.e. check if $s \in \text{Reachable } M_{RCV}$ then $p \in L_{RCV}(s)$, i.e.:
    \[
    \begin{align*}
    p &\in L_{RCV}(111) \land \\
    p &\in L_{RCV}(011) \land \\
    p &\in L_{RCV}(000) \land \\
    p &\in L_{RCV}(100) \land \\
    p &\in L_{RCV}(010) \land \\
    p &\in L_{RCV}(110)
    \end{align*}
    \]

- Example
  - if $AP = \{A, B\}$
  - and $L_{RCV}(s) = \text{ if } s \in \{001, 101\} \text{ then } \{A\} \text{ else } \{B\}$
  - then $M_{RCV} \models AG\ A$ is not true, but $M_{RCV} \models AG\ B$ is true
Symbolic Boolean model checking of reachability

- Assume states are \( n \)-tuples of Booleans \((b_1, \ldots, b_n)\)
  - \( b_i \in \mathbb{B} = \{\text{true, false}\} (= \{1, 0\}) \)
  - \( S = \mathbb{B}^n \), so \( S \) is finite: \( 2^n \) states

- Assume \( n \) distinct Boolean variables: \( v_1, \ldots, v_n \)
  - e.g. if \( n = 3 \) then could have \( v_1 = x, v_2 = y, v_3 = z \)

- Boolean formula \( f(v_1, \ldots, v_n) \) represents a subset of \( S \)
  - \( f(v_1, \ldots, v_n) \) only contains variables \( v_1, \ldots, v_n \)
  - \( f(b_1, \ldots, b_n) \) denotes result of substituting \( b_i \) for \( v_i \)
  - \( f(v_1, \ldots, v_n) \) determines \( \{(b_1, \ldots, b_n) \mid f(b_1, \ldots, b_n) \iff \text{true}\} \)

- Example \( \neg(x = y) \) represents \( \{(\text{true, false}), (\text{false, true})\} \)

- Transition relations also represented by Boolean formulae
  - e.g. \( R_{RCV} \) represented by:
    \[
    (q0' = dreq) \land (dack' = (dreq \land (q0 \lor (\neg q0 \land dack))))
    \]
Symbolically represent Boolean formulae as BDDs

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

- Variables:
  $v = \text{if } v \text{ then } 1 \text{ else } 0$
  $\neg v = \text{if } v \text{ then } 0 \text{ else } 1$

- Example: BDDs of variable $v$ and $\neg v$

- Example: BDDs of $v_1 \land v_2$ and $v_1 \lor v_2$
More BDD examples

- **BDD of** $v_1 = v_2$

- **BDD of** $v_1 \neq v_2$
BDD of a transition relation

- BDDs of

\[(v_1' = (v_1 = v_2)) \land (v_2' = (v_1 \neq v_2))\]

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 $f_1 \land f_2, f_1 \lor f_2$ 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 $\exists \vec{u}. f(\vec{u}) \land g(\vec{u}, \vec{v})$ represents $\{ \vec{v} \mid \exists \vec{u}. \vec{u} \in S \land R \vec{u} \vec{v} \}$

- 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 $h(\top, v_2) \lor h(\bot, v_2)$

- From BDD of formula $f(v_1, \ldots, v_n)$ can compute $b_1, \ldots, b_n$ such that if $v_1 = b_1, \ldots, v_n = b_n$ then $f(b_1, \ldots, b_n) \leftrightarrow \text{true}$
  - $b_1, \ldots, b_n$ is a satisfying assignment (SAT problem)
  - used for counterexample generation (see later)