A property not expressible in LTL

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

- $M = (\{s_0, s_1\}, \{s_0\}, \{(s_0, s_0), (s_0, s_1), (s_1, s_1)\}, L)$
- $M' = (\{s_0\}, \{s_0\}, \{(s_0, s_0)\}, L)$

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

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

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

- For LTL path formulae $\phi$ recall that $M \models \phi$ is defined by:
  \[
  M \models \phi \iff \forall \pi \text{ s. } s \in S_0 \land \text{Path } R s \pi \Rightarrow \llbracket \phi \rrbracket_M(\pi)
  \]

- For CTL state formulae $\psi$ the definition of $M \models \psi$ is:
  \[
  M \models \psi \iff \forall s. \text{ s } S_0 \Rightarrow \llbracket \psi \rrbracket_M(s)
  \]

- $M$ common; LTL, CTL formulae and semantics $\llbracket \cdot \rrbracket_M$ differ

- CTL model checking algorithm:
  - compute $\{ s \mid \llbracket \psi \rrbracket_M(s) = true \}$ bottom up
  - check $S_0 \subseteq \{ s \mid \llbracket \psi \rrbracket_M(s) = true \}$
  - symbolic model checking represents these sets as BDDs
CTL model checking: $p$, $\text{AX} \psi$, $\text{EX} \psi$

- For CTL formula $\psi$ let $\{\psi\}_M = \{s \mid [\psi]_M(s) = \text{true}\}$
- 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 $\{\text{AX} \psi\}$
  - recursively compute $\{\psi\}$
  - marks those states all of whose successors are in $\{\psi\}$
  - $\{\text{AX} \psi\}$ is the set of marked states

- To compute $\{\text{EX} \psi\}$
  - recursively compute $\{\psi\}$
  - marks those states with at least one successor in $\{\psi\}$
  - $\{\text{EX} \psi\}$ is the set of marked states
CTL model checking: \( \{E[\psi_1 U \psi_2]\}, \{A[\psi_1 U \psi_2]\}\)

- To compute \( \{E[\psi_1 U \psi_2]\}\)
  - 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
  - \( \{E[\psi_1 U \psi_2]\}\) is set of marked states

- More formally: \( \{E[\psi_1 U \psi_2]\}\) = \( \bigcup_{n=0}^{\infty} \{E[\psi_1 U \psi_2]\}_n \) where:
  - \( \{E[\psi_1 U \psi_2]\}_0 = \{\psi_2\} \)
  - \( \{E[\psi_1 U \psi_2]\}_{n+1} = \{E[\psi_1 U \psi_2]\}_n \cup \{s \in \{\psi_1\} | \exists s' \in \{E[\psi_1 U \psi_2]\}_n. R s s'\} \)

- \( \{A[\psi_1 U \psi_2]\}\) similar, but with a more complicated iteration
  - details omitted (see Huth and Ryan)
Example: checking $\textbf{EF} \, p$

- $\textbf{EF} \, p = \textbf{E}[\textbf{T} \, \textbf{U} \, p]$
  - holds if $\psi$ holds along some path

- Note $\{\textbf{T}\} = S$

- Let $S_n = \{\textbf{E}[\textbf{T} \, \textbf{U} \, p]\}_n$ then:
  
  $S_0 = \{\textbf{E}[\textbf{T} \, \textbf{U} \, p]\}_0$
  $= \{p\}$
  $= \{s \mid p \in L(s)\}$

  $S_{n+1} = S_n \cup \{s \in \{\textbf{T}\} \mid \exists s' \in \{\textbf{E}[\textbf{T} \, \textbf{U} \, p]\}_n. \, R \, s \, s'\}$
  $= S_n \cup \{s \mid \exists s' \in S_n. \, R \, s \, s'\}$

- mark all the states labelled with $p$
- mark all with at least one marked successor
- repeat until no change
- $\{\textbf{EF} \, p\}$ 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:

\[
M = (S_{RCV}, S_{0_{RCV}}, R_{RCV}, L_{RCV})
\]

and

\[
R_{RCV} (dreq, q0, dack) (dreq', q0', dack') = (q0' = dreq) \land (dack' = (dreq \land (q0 \lor dack)))
\]
Possible states for RCV:
\{000, 001, 010, 011, 100, 101, 110, 111\}
where \(b_2 b_1 b_0\) denotes state
\[
dreq = b_2 \land q0 = b_1 \land dack = b_0
\]
Graph of the transition relation:
Computing \textbf{Reachable} $M_{RCV}$

Define:

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

\[
S_{i+1} \quad = \quad S_i \cup \{ s' \mid \exists s \in S_i. \ R_{RCV} \ s \ s' \}
= \quad S_i \cup \{ b'_2 b'_1 b'_0 \mid \\
\exists b_2 b_1 b_0 \in S_i. (b'_1 = b_2) \land (b'_0 = b_2 \land (b_1 \lor b_0))\}
\]
Computing $\{\text{EF } \text{At111}\}$ where $\text{At111} \in L_{\text{RCV}}(s) \iff s = 111$

Define:

$S_0 = \{s \mid \text{At111} \in L_{\text{RCV}}(s)\}$

$= \{s \mid s = 111\}$

$= \{111\}$

$S_{n+1} = S_n \cup \{s \mid \exists s' \in S_n. \ R(s, s')\}$

$= S_n \cup \{b_2b_1b_0 \mid \exists b'_2b'_1b'_0 \in S_n. (b'_1 = b_2) \land (b'_0 = b_2 \land (b_1 \lor b_0))\}$
Computing \( \{\text{EF At111}\} \) (continued)

Compute:

\[
\begin{align*}
S_0 &= \{111\} \\
S_1 &= \{111\} \cup \{101, 110\} \\
&= \{111, 101, 110\} \\
S_2 &= \{111, 101, 110\} \cup \{100\} \\
&= \{111, 101, 110, 100\} \\
S_3 &= \{111, 101, 110, 100\} \cup \{000, 001, 010, 011\} \\
&= \{111, 101, 110, 100, 000, 001, 010, 011\} \\
S_n &= S_3 \quad (n > 3) \\
\end{align*}
\]

\(\{\text{EF At111}\} = \mathbb{B}^3 = S_{\text{RCV}}\)

\(M_{\text{RCV}} \models \text{EF At111} \iff S_{0_{\text{RCV}}} \subseteq S\)
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 \( \{\neg\psi\} \), \( \{\psi_1 \land \psi_2\} \), \( \{\psi_1 \lor \psi_2\} \), \( \{\psi_1 \Rightarrow \psi_2\} \) computed using standard BDD algorithms
  - BDDs of \( \{AX\psi\} \), \( \{EX\psi\} \), \( \{A[\psi_1 U \psi_2]\} \), \( \{E[\psi_1 U \psi_2]\} \) 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](http://www.cs.cmu.edu/~modelcheck/smv.html) (original)
  - [http://nusmv.irst.itc.it/](http://nusmv.irst.itc.it/) (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
  
  \[
  \begin{array}{c}
  0: & P := 1; \\
  s0 & 1: \text{WHILE } (\star) \text{ DO SKIP;} \\
  s1 & 2: P := 0; \\
  s2 & 3: P := 1; \\
  4: & \text{WHILE } T \text{ DO SKIP;} \\
  5: & 
  \end{array}
  \]

- Property true, but cannot be expressed in CTL
  - would need something like \( \text{AF} \psi \)
  - where \( \psi \) is something like "property \( p \) true from now on"
  - but in CTL \( \psi \) must start with a path quantifier \( A \) or \( E \)
  - cannot talk about current path, only about all or some paths
  - \( \text{AF} (\text{AG} \ p) \) is false (consider path \( s0 s0 s0 \cdots \))
LTL can express things CTL can’t

- Recall:
  \[
  \begin{align*}
  \llbracket F \phi \rrbracket_M(\pi) &= \exists i. \llbracket \phi \rrbracket_M(\pi \downarrow i) \\
  \llbracket G \phi \rrbracket_M(\pi) &= \forall i. \llbracket \phi \rrbracket_M(\pi \downarrow i)
  \end{align*}
  \]

- \(FG\phi\) is true if there is a point after which \(\phi\) is always true
  \[
  \begin{align*}
  \llbracket FG\phi \rrbracket_M(\pi) &= \llbracket F(G(\phi)) \rrbracket_M(\pi) \\
  &= \exists m_1. \llbracket G(\phi) \rrbracket_M(\pi \downarrow m_1) \\
  &= \exists m_1. \forall m_2. \llbracket \phi \rrbracket_M((\pi \downarrow m_1) \downarrow m_2) \\
  &= \exists m_1. \forall m_2. \llbracket \phi \rrbracket_M(\pi \downarrow (m_1 + m_2))
  \end{align*}
  \]

- LTL can express things that CTL can’t express

- Note: it’s tricky to prove CTL can’t express \(FG\phi\)
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 57)

- 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
Both state formulae ($\psi$) and path formulae ($\phi$)

- state formulae $\psi$ are true of a state $s$ like CTL
- path formulae $\phi$ are true of a path $\pi$ like LTL

Defined mutually recursively

\[
\psi ::= \begin{aligned}
\ p & \quad \text{(Atomic formula)} \\
\ \neg \psi & \quad \text{(Negation)} \\
\ \psi_1 \lor \psi_2 & \quad \text{(Disjunction)} \\
\ A\phi & \quad \text{(All paths)} \\
\ E\phi & \quad \text{(Some paths)}
\end{aligned}
\]

\[
\phi ::= \begin{aligned}
\ \psi & \quad \text{(Every state formula is a path formula)} \\
\ \neg \phi & \quad \text{(Negation)} \\
\ \phi_1 \lor \phi_2 & \quad \text{(Disjunction)} \\
\ X\phi & \quad \text{(Successor)} \\
\ F\phi & \quad \text{(Sometimes)} \\
\ G\phi & \quad \text{(Always)} \\
\ [\phi_1 \ U \phi_2] & \quad \text{(Until)}
\end{aligned}
\]

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

- Combines CTL state semantics with LTL path semantics:

\[
\begin{align*}
[p]_M(s) &= p \in L(s) \\
[\neg \psi]_M(s) &= \neg([\psi]_M(s)) \\
[\psi_1 \lor \psi_2]_M(s) &= [\psi_1]_M(s) \lor [\psi_2]_M(s) \\
[A\phi]_M(s) &= \forall \pi. \text{Path } R \ s \ \pi \Rightarrow \phi(\pi) \\
[E\phi]_M(s) &= \exists \pi. \text{Path } R \ s \ \pi \land [\phi]_M(\pi) \\
[\psi]_M(\pi) &= [\psi]_M(\pi(0)) \\
[\neg \phi]_M(\pi) &= \neg([\phi]_M(\pi)) \\
[\phi_1 \lor \phi_2]_M(\pi) &= [\phi_1]_M(\pi) \lor [\phi_2]_M(\pi) \\
[X\phi]_M(\pi) &= [\phi]_M(\pi\downarrow 1) \\
[F\phi]_M(\pi) &= \exists m. [\phi]_M(\pi\downarrow m) \\
[G\phi]_M(\pi) &= \forall m. [\phi]_M(\pi\downarrow m) \\
[[\phi_1 U \phi_2]]_M(\pi) &= \exists i. [\phi_2]_M(\pi\downarrow i) \land \forall j. j<i \Rightarrow [\phi_1]_M(\pi\downarrow j)
\end{align*}
\]

- Note \([\psi]_M : S \rightarrow B\) and \([\phi]_M : (N \rightarrow S) \rightarrow B\)
LTL and CTL as CTL*

- As usual: \( M = (S, S_0, R, L) \)
- If \( \psi \) is a CTL* state formula: \( M \models \psi \iff \forall s \in S_0. \llbracket \psi \rrbracket_M(s) \)
- If \( \phi \) is an LTL path formula then: \( M \models_{\text{LTL}} \phi \iff M \models_{\text{CTL}^*} A\phi \)
- If \( R \) is total (\( \forall s. \exists s'. R s s' \)) then (exercise):
  \( \forall s s'. R s s' \iff \exists \pi. \text{Path } R \ s \ \pi \land (\pi(1) = s') \)
- The meanings of CTL formulae are the same in CTL*:

\[
\llbracket A(X \psi) \rrbracket_M(s) = \forall \pi. \ \text{Path } R \ s \ \pi \Rightarrow \llbracket X \psi \rrbracket_M(\pi) = \forall \pi. \ \text{Path } R \ s \ \pi \Rightarrow \llbracket \psi \rrbracket_M(\pi)(1)
\]

\[
\llbracket AX \psi \rrbracket_M(s) = \forall s'. R s s' \Rightarrow \llbracket \psi \rrbracket_M(s') = \forall s'. (\exists \pi. \ \text{Path } R \ s \ \pi \land (\pi(1) = s')) \Rightarrow \llbracket \psi \rrbracket_M(s')
\]

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 $G(Fp) \Rightarrow G(Fq)$

- Can’t say this in CTL
  - why not – what’s wrong with $AG(AFp) \Rightarrow AG(AFq)$?
  - in CTL* expressible as $A(G(Fp) \Rightarrow G(Fq))$
  - fair CTL model checking implemented in checking algorithm
  - fair LTL just a fairness assumption like $G(Fp) \Rightarrow \cdots$

- Fairness is a tricky and subtle subject
  - many kinds of fairness:
    ‘weak fairness’, ‘strong fairness’ etc
  - exist whole books on fairness
Propositional modal $\mu$-calculus

- You may learn this in *Topics in Concurrency*

- $\mu$-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 $\mu$-calculus

- Strictly stronger than CTL*
  - expressibility strictly increases as allowed nesting increases
  - need fixed point operators nested 2 deep for CTL*

- The $\mu$-calculus is very non-intuitive to use!
  - intermediate code rather than a practical property language
  - nice meta-theory and algorithms, but terrible usability!