Logic-Based Models & Tools

Temporal Logic

- Temporal logic has a certain appeal for real-time (and concurrency).
- The standard “temporal logics” might more properly be called “sequence logics”, since they are based on sequences of events or discrete time steps.
- Metric temporal logic treats time as a continuous-valued quantity.

Temporal Logic Operators

- □ means “henceforth”:
  If P is a predicate on state, then □P means that P will be true from now on.
- ◇ means “eventually”:
  ◇P means that P will be true in some “future” state (starting with now).

Now

- In the absence of temporal operators, a predicate is understood to be true “now”.
- Since “now” is arbitrary, this is like putting a □ in front of the predicate (similar to the way in which unquantified variables are implicitly ∀-quantified).

TL Examples

- (□ switch on) ⇒ (◇ motor running)
- motor running forward ⇒ (◇ switch off)
- □¬(p in critical section ∧ q in critical section)
- p at entrance to critical section ⇒ ◇p in critical section
- p in critical section ⇒ x = 1
- req = 1 ⇒ ◇p is bus master

Safety & Liveness

- These are two complementary kinds of properties:
  - Safety: □¬B means that “nothing bad will happen”
  - Liveness: ◇G means that “something good will happen”
**Connection between \(\Box\) and \(\Diamond\)**

- \(\Box P = \neg \Diamond \neg P\)
- \(\Diamond P = \neg \Box \neg P\)
- sort of like the relationships between \(\forall\) and \(\exists\) (deMorgan’s law)

**Idioms and Reductions**

- \(\Diamond \Box P = P\) is true infinitely often
- \(\Box \Diamond P = P\) will eventually be true forever
- \(\Diamond \Diamond P = \Diamond P\)
- \(\Box \Box P = \Box P\)

**Other Operators**

- \(\Diamond P = P\) is true in the “next” state
- \(\Box P = P \land \Diamond \Box P\)
- \(\Diamond P = P \lor \Diamond \Diamond P\)
- \(P \cup Q\) (P until Q) = \((\Diamond Q) \land \neg Q) \Rightarrow P\)
- \(P \circ Q\) (P since Q) = \(Q \Rightarrow \Box P\)

**Time Rover**

- Time Rover, aka Temporal Rover is a commercial specification product based on temporal logic.
- The user formulates temporal logic assertions and Time Rover checks them in the context of live data.
- Time Rover will also generate test sequences for the system.

**Time Rover Strategy**

![Diagram showing temporal logic strategy and examples](http://www.time-rover.com/timing1.html)

**Examples from Time Rover**

![Example assertions and temporal logic strategies](http://www.time-rover.com/timing1.html)
Examples from Time Rover
(http://www.time-rover.com/timing2.html)

Always ackDma Implies (Not ackCpu Since reqDMA)

\[ (\text{ackDma}) \Rightarrow (\neg \text{ackCpu} \land \text{reqDMA}) \]

"Always, when bus is granted to DMA, there should be an earlier time with a DMA bus request that was not followed by a CPU request."

Examples from Time Rover

Always ackDma Implies (Eventually reqCpu Since reqDMA)

\[ (\text{ackDma}) \Rightarrow (\text{reqCpu} \land \text{reqDMA}) \]

"Always, when bus is granted to DMA, there should be an earlier time with a DMA bus request that was followed by a CPU request."

Real Time Constraint Spec.
(http://www.time-rover.com/timing1rt.html)

Always ackDma Implies (Not ackCpu Until<20 eop)

\[ (\text{ackDma}) \Rightarrow (\neg \text{ackCpu} \land <20 \text{eop}) \]

"Always, if bus is granted to DMA, it should not be granted to CPU until eop, and eop must occur within 20 time units."

Priority Inversion Spec., re Mars Pathfinder
(http://www.time-rover.com/Priority.html)

Let

- HIGHPriorityTaskBlocked() represent a situation where the information bus thread is blocked by the low priority meteorological data gathering task.
- HIGHPriorityTaskInMutex() represent a situation where the information bus thread is in Mutex.
- LOWPriorityTaskInMutex() represent a situation where the meteorological thread is in Mutex.
- MEDPriorityTaskRunning() represent a situation where the communications task is running.

Want

- Not Eventually (HIGHPriorityTaskBlocked() And MEDPriorityTaskRunning())
- Always (LOWPriorityTaskInMutex()) Implies Not (MEDPriorityTaskRunning()) Until (HIGHPriorityTaskInMutex())

From Time Rover Web Page
(http://www.time-rover.com/Priority.html)

"Using such assertions (written as comments in the Pathfinder code), the Temporal Rover would generate code that announces success and/or failure of any assertion during testing.

Interestingly enough, the JPL engineers actually created a priority inversion situation during testing, but did not manage to analyze their recorded data well enough so to conclude that priority inversion is indeed a bug in their system. In other words, their test runs were sufficient, but their analysis tools were not."

ATM Example: Sequence Diagram
### ATM Example: Time Rover Specs

- **Always (\{\text{InsertCard}\} \implies \text{Eventually (\text{ATMcashDispensed})} \lor \text{ATMErrorDisplay}) )**

- **Always (\{\text{ConsortiumSuccess}\} \implies (\text{Not} \{\text{InsertCard}\}) \lor \text{ATMcashDispensed})**

- **Always (\{\text{ConsortiumSuccess}\} \implies \text{Eventually (\text{ATMcashDispensed})})**

- **Always (\{\text{ConsortiumSuccess}\} \implies \text{Eventually (C1<30 \text{ATMcashDispensed})})**

### Other Generic Time Rover Specs

(http://www.time-rover.com/using.html)

- ev1 and ev2 happen or do not happen simultaneously: Always (ev1 \iff ev2)
- if ev1 then ev2 two cycles later: Always (ev1 \implies \text{Eventually (2)(ev2)})
- ev2 not before ev1: Always (Not(ev2) \Until ev1)
- ev2 within n cycles after ev1: Always (ev1 \implies \text{Eventually \leq n(ev2)})
- ev2 within m and n cycles after ev1: Always (ev1 \implies \text{Eventually ([m,n](ev2)})
- ev2 any number of cycles after ev1: Always (ev1 \implies \text{Eventually (ev2)})
- ev2 after n cycles of no ev1: Always (ev1 \implies \text{Eventually (ev2)})
- ev2 any number of cycles after ev1, with no ev3 in between: Always (ev1 \implies (\text{Not}(ev3) \Until (ev2)) \And \text{Eventually (ev2)})

### Other Generic Specs

- ev1 after the last ev2 and before ev3:
  - Always (Last(ev2) \implies \text{Eventually (ev1)} \And \text{Always(InThePast (Not(ev3))})
- if ev1, then ev2 must not occur for n cycles:
  - Always (ev1 \implies \text{Always \leq n(Not(ev2))})
- if ev2, then ev1 must have occurred 3 cycles earlier:
  - (ev2) \implies \text{Previous \leq 2(Previous (ev1))}
- If ev2, then ev1 must have occurred sometime earlier (not including present time): Always (\{ev2\} \implies \text{Previous SometimeInThePast (ev1)})
- ev2/ev1 occurred at least once between evStart and evStop:
  - Always (ev1 \implies \text{Previous (evStart)} \And \text{Previous (evStop)})
- val=VAL between evStart and evStop:
  - Always (\{val=VAL\} \Until (evStop) \Or Always (\{val=VAL\} \Between (evStart), (evStop)))

### CTL (Computation Tree Logic)

- CTL, a form of temporal logic, is the basis for a formal analysis implementation.
- http://www.cs.bham.ac.uk/research/lics/

### Models for CTL

- The intention is that CTL formulas apply to a non-deterministic state-transition model, in which certain propositions evaluate to true or false in every state.

### Example Model for CTL

![Example Model](p, q, r are atomic propositions, shown in the states for which they are true.)
Graphs can be “Unwound” into Trees

Technical Requirement
- Every state has at least one transition out of it (although that transition may immediately return).

Technical Requirement
- Every state has at least one transition out of it (although that transition may immediately return).

CTL Operators
- CTL expresses formulas about the model.
- A formula can make assertions about a single state or about paths from a state.
- The usual propositional operators are included in formulas, as are temporal operators described next.

CTL Temporal Operators
- Relative to a given state and the paths leaving it:
  - $\mathbf{AX} p$: All Next states satisfy $p$.
  - $\mathbf{AG} p$: Along all paths, all states satisfy $p$.
  - $\mathbf{AF} p$: Along all paths, some state satisfies $p$.
  - $\mathbf{A}[p U q]$: Along all paths, $p$ holds until $q$.
  - $\mathbf{EX} p$: Some Next state satisfies $p$.
  - $\mathbf{EG} p$: Along some path, all states satisfy $p$.
  - $\mathbf{EF} p$: Along some path, some state satisfies $p$.
  - $\mathbf{E}[p U q]$: Along some path, $p$ holds until $q$.
- [$A = \text{All}, E = \text{Exists}, X = \text{next}, G = \text{Global}, F = \text{Future}, U = \text{Until}$]
- Note that “future” includes the present.

Comparison with Classical Temporal Operators
- $\mathbf{AX} p$ is like $\Box p$
- $\mathbf{AG} p$ is like $\square p$
- $\mathbf{AF} p$ is like $\Diamond p$
- $\mathbf{A}[p U q]$ is like $p U q$

AX $p$ (All neXt)
**EX p (Some Next)**

**AG p (All Global)**

Note: For “future” includes the present.

**EG p (Some Global)**

**AF p (All Future)**

Note: For “future” includes the present.

**EF p (Some Future)**

**A[p U q] (All … Until)**

Note: For “Until”, q is required to hold in some future state.

Note: For “future” includes the present.
Note: For “Until”, q is required to hold in some future state.

Typical Formulas of Interest in Concurrent Systems

- AG (requested ⇒ AF granted)
- AG (AF enabled) [enabled infinitely-often]
- AF (AG deadlocked) [will always deadlock]
- A[motor-on U switch-off]

Model Checking (Clarke and Emerson)

(output: 
- yes
- no + counterexample)

Labeling Algorithm

- The labeling algorithm is used for model-checking.
- Given a finite-state model and a formula, the algorithm will determine:
  - whether the formula is true for the model
  - if not, a state sequence that refutes the formula
Adequacy

- In order to simplify the algorithm, it is desirable to work with as few operators as possible.
- A set S of operators is adequate if the other operators can be expressed using only operators in S.

Adequate Operators

- Propositional: $\land$, $\neg$, $\bot$ ($\bot$ = "false")
- Temporal: AF, EU, EX
- Examples of translation:
  - $T = \neg \bot$
  - $p \lor q = \neg (\neg p \land \neg q)$
  - $\neg \text{AX}\phi = \text{EX} \neg \phi$
  - $\text{EG}\phi = \neg \text{AF} \neg \phi$
  - $\text{EF}\phi = \text{E}[T \lor \phi]$
  - $\text{A}[p \lor q] = \neg (\text{E}[\neg \phi \lor (\neg p \land \neg q)] \lor \text{EG} \neg \phi)$

Labeling Algorithm

- Input: A finite-state model, consisting of states and transitions, as well as a function $L$: states $\rightarrow$ proposition symbols and a CTL formula $\phi$.
- Output: The set of states of the model for which the formula $\phi$ is true.

Labeling Algorithm (1), by Sub-Formulas of $\phi'$

- $\bot$: label no states with this formula.
- $p$ (propositional variable): label with $p$ those states $s$ that have $p \in L(s)$.
- $\psi_1 \land \psi_2$: label with $\psi_1 \land \psi_2$ those states that are labeled with both $\psi_1$ and $\psi_2$.
- $\neg \psi$: label with $\neg \psi$, those states that are not labeled with $\psi$.
- $\text{EX}\psi$: label with $\text{EX}\psi$, those states having at least one successor labeled $\psi$.

Labeling Algorithm (2), by Sub-Formulas of $\phi'$

- $\text{AF}\psi$: label with $\text{AF}\psi$ (recursively):
  - all states labeled with $\psi$.
  - all states the successors of which are labeled with $\text{AF}\psi$.

\[\text{AF}\psi\quad\text{AF}\psi\]

\[\text{AF}\psi\quad\text{AF}\psi\]

\[\text{AF}\psi\quad\text{AF}\psi\]
Labeling Algorithm (2), by Sub-Formulas of $\phi'$

- $E[\psi_1 U \psi_2]$: label with $E[\psi_1 U \psi_2]$ (recursively):
  - all states labeled with $\psi_2$
  - any state labeled with $\psi_2$ and having a successor labeled with $E[\psi_1 U \psi_3]$.

Example: Show: $\forall G T_1 \Rightarrow \forall F C_1$

Transformation Steps

- To be shown: $\forall G T_1 \Rightarrow \forall F C_1$
- $\forall G \psi$ is true for a state iff $\psi$ is true for all states reachable from that state.
- For simplicity, we work with: $T_1 \Rightarrow \forall F C_1$
- Transform to: $\neg (T_1 \land \neg \forall F C_1)$
- We'll start labeling with: $T_1$ and $C_1$

First-Level Labelings

Second-Level Labeling-Basis

Second-Level Labeling-Recursion
Third-Level Labelings

Fourth-Level Labeling ($\neg(T_1 \land \neg \text{AF } C_1)$)

(Fifth-Level Labeling ($\neg(T_1 \land \neg \text{AF } C_1)$)

(SMV (Symbolic Model Verifier)
- SMV is a “modeling language” for systems.
- It resembles a programming language, but includes non-determinism.
- It includes a model checker for CTL formulas.
- Original was developed at CMU.
- Cadence version is available for Windows on the web:
  http://www-cad.eecs.berkeley.edu/~kenmcmil/

SMV Spec for a Mutex Problem

SMV Example

SMV spec:

```
MODULE main
VAR
  request : boolean;
  status : {ready,busy} ;
ASSIGN
  init(status) := ready;
  next(status) :=
    case
    request : busy;
    1 : {ready,busy};
    esac;
SPEC
  AG(request -> AF status = busy)
```

Finite-state model:

```
MODULE main
VAR
  request : boolean;
  status : {ready,busy} ;
ASSIGN
  init(status) := ready;
  next(status) :=
    case
    request : busy;
    1 : {ready,busy};
    esac;
SPEC
  AG(request -> AF status = busy)
```

Busy is shorthand for status = busy and ready is for: status = ready

http://www.cs.bham.ac.uk/research/lics/ancillary/smv/index.html also gives code for the alternating-bit protocol.
SMV on turing

- For examples, see: /cs/cs156/smv/doc/smv/examples
- To set up:
  - setenv SMV_DIR /cs/cs156/smv
  - setenv PATH $SMV_DIR/bin:$PATH
  - setenv MANPATH $SMV_DIR/man:$MANPATH
  - setenv LD_LIBRARY_PATH $SMV_DIR/lib:$LD_LIBRARY_PATH