Temporal Logic in Rosetta

Kylie Williams

Overview of Temporal Logic

- Temporal Combinators:
  - X – if P holds in this state then X P states that P must hold in the next state.
  - F – F P states that the property P must hold in some future state.
  - G – G P states that the property P must hold in all future states.
  - U – U P1 U P2 states that P1 holds until P2 holds.

- Path Operators
  - A – A P states that all executions out of the current state satisfy P.
  - E – E P states that from the current state, there exists an execution satisfying P.
  - The path operators (A & E) and temporal operators (G & F) are often used in pairs.
Temporal Logic in Rosetta
Kripke Structures

- A Kripke structure has states, transitions and a function that labels each state with a set of properties that are true in that state.
- Paths in a Kripke structure model computations of the system.

Kripke Domain

domain kripke::static is
S:: univ, // set of states
S0:: subset(s); // set of initial states
R:: subset(s*s); // set of transitions
trace :: sequence(S);

Begin
T1: forall(s::S | exists(s1::S | (s,s1) in R));
T2: forall(trace(k) :: S | (trace(k), trace(k+1)) in R);
End kripke;
### CTL* Domain extends Kripke

```plaintext
property :: <* (arg:: univ) :: boolean *>; 
X_ :: <* (rhs :: property) :: property *>; 
F_ :: <* (rhs :: property) :: property *>; 
G_ :: <* (rhs :: property) :: property *>; 
__U_ :: <* (lhs, rhs :: property) :: property *>; 
__R_ :: <* (lhs, rhs :: property) :: property *>; 
A_ :: <* (rhs :: property) :: property *>; 
E_ :: <* (rhs :: property) :: property *>; 
satisfied_by(M :: kripke, e :: trace, k :: property) :: boolean; 
satisfied_by(M :: kripke, s :: S, k :: property) :: boolean; 
```

05/05/2005  
Kylie Williams  
7

### Operator Axioms

1. T1: satisfied_by(M, e, X k) = satisfied_by(M, tail(e), k);
   T2: satisfied_by(M, e, F k) = exists(t :: int | t >= 0 and 
   satisfied_by(M, e sub [e(t),..#(e-1)], k));
   T3: satisfied_by(M, e, G k) = forall(t :: int | t >= 0 and satisfied_by(M, 
   e sub [e(t),..#(e-1)], k)); 
   T4: satisfied_by(M, e, k U l) = exists(a :: int | a >= 0 and 
   satisfied_by(M, e sub [e(a),..#(e-1)], l) and forall(b :: int | b >= 0 
   and b < a and satisfied_by(M, e sub [e(b),..#(e-1)], k)); 
   T5: satisfied_by(M, S, E k) = exists(e :: trace | satisfied_by(M, e, k)); 
   T6: satisfied_by(M, S, A k) = forall(e :: trace | satisfied_by(M, e, k));