CSEP590 – Model Checking and Automated Verification

Lecture outline for July 23, 2003

-Today, we will talk about a few "loose ends" from previous lectures, as well as model checking for timed, reactive systems.
-First, we deal with Fairness in model checking
    -$M,s_0 \models \phi$ may fail due to unrealistic behavior
        -Example: 2 processes with critical sections. Process1 may stay indefinitely in critical section, preventing Process2 from every entering its critical section.
    -Fairness constraints: state that a given formula is true infinitely often on every computation path
        -Such paths are fair computation paths
    -How accomplish? When evaluating truth of CTL formula, A and E connectives only range over fair paths
    -Defn: Let $C = \{f_1, f_2, \ldots f_n\}$ be a set of n fairness constraints. A computation path $s_0 \rightarrow s_1 \rightarrow$ is fair with respect to C if for each i there are infinitely many j s.t. $s_j \models f_i$, that is, each f_i is true infinitely often along the path

-We'll let A_C and E_C denote the operations A and E restricted to fair paths

-Recall: EU, EG, and EX form an adequate set for CTL

  -Therefore, E_CU, E_CG, and E_CX form an adequate set for fair CTL

  -Indeed, E_CU and E_CX can be represented in terms of E_CG, thus we only need an algorithm for checking $E_CG\phi$:

    -Restrict graph to states satisfying ϕ

      -In this graph, want to know from which states there is a fair computation path

    -Find the maximal SCCs (Strongly Connected Components) of restricted graph

    -Remove a SCC is for some f_i, it doesn't contain a state satisfying f_i. Result SCCs are "fair SCCs"

      -Any state of restricted graph that can reach a fair SCC has a fair path from it

    -Use search to find such states

-The complexity of this algorithm is O(n*f*(V+E)) => still linear!

-Extensions and Alternatives to CTL

-Linear Time Logic (LTL)

-Close to CTL, but formulas have meanings on individual computation paths => no quantifiers A and E

-Is LTL less expressive than CTL? More expressive?

-LTL syntax for a formula ϕ

-ϕ := p | (! ϕ) | (ϕ and ϕ) | (ϕ U ϕ) | (Gϕ) | (Fϕ) | (Xϕ)

-Formula is evaluated on a path or a set of paths

-Set of paths satisfy formula if every path in the set does

-Consider path $\pi = s_1 \rightarrow s_2 \rightarrow \ldots$ where π^i represents the suffix starting at s_i

-Defn: give a model M for CTL, define when a path π satisfies an LTL formula via |= relation:

-1) $\pi \models T$

-2) $\pi \models p$ iff p is in $L(s_1)$

-3) $\pi \models !\phi$ iff $\pi \not\models \phi$

-4) $\pi \models \phi_1$ and ϕ_2 iff $\pi \models \phi_1$ and $\pi \models \phi_2$

-5) $\pi \models X\phi$ iff $\pi^2 \models \phi$

-6) $\pi \models G\phi$ iff for all i at least 1, $\pi^i \models \phi$

-7) $\pi \models F\phi$ iff for some i at least 1, $\pi^i \models \phi$

-8) $\pi \models \phi_1 U \phi_2$ iff for some i at least 1 s.t. $\pi^i \models \phi_2$ and for all j = 1…i-1 we have $\pi^j \models \phi_1$

-LTL formula is satisfied in a state s of the model if the formula is satisfied on every path starting at s

-LTL has the usual G and F equivalences, as well as distribution over AND and OR

-There is also 1 very important equivalence we will see, which is relied upon to show that EG, EU, EX form an adequate set

-CTL* - allows nested modalities and boolean connectives before applying path quantifiers E and A.

-We'll see some examples of this in class

-Syntax of CTL*

-Divides formulas into 2 classes

-State formulas: evaluated in states:

-ϕ := p | T | ! ϕ | (ϕ and ϕ) | A[α] | E[α]

-Path formulas: evaluated along paths:

-α := ϕ | ! α | (α and α) | (αU α) | G α | F α | X α

-This is a mutually recursive definition

-LTL us a subset of CTL*. Why?

-CTL is subset of CTL*. Why?

-We'll see in class examples of formulas that define the differences between these 3 logics

-Timed Automata

-Model reactive systems where there are notions of “real-time”

-Ex: “trigger the alarm upon detection of a problem” vs. “trigger the alarm in less than 5 seconds after detecting the problem”

-How doe we model such systems? How do we verify them?

-We’ve seen one way: basic synchronization based on a global clock

-Very inadequate though

-Timed Automata – model quantitative info on passage of time

-2 elements:

-Finite automata

-Clocks (associated with transitions)

-Take on non-negative real values

-All clocks start out null in the initial state

-A configuration of the system is (q,v) where q is the current control state and v is a valuation of the automaton's clocks

-Configurations change in 1 of 2 ways

-A delay d in time elapses, in which case all clocks are updated by d ((q,v) → (q, v+d))

-Discrete transition – an action transition (as with normal automata, a control state change). Some clocks may be reset to 0 on such transitions

-We'll see an example in class

-Networks of Timed Automata

-Composite model composed of many timed automata synchronized.

-All clocks across all components are updated on delays

-Similar to what we saw with modeling systems via automata

-Example in class: the classical railway example

-There are 3 common extensions to this model of timed automata

-Invariants: guarantee that a certain transition eventually occurs by placing invariants on clocks in a state

-If no transition is taken, invariants expire and system reaches deadlock

-Urgency: transition that can't tolerate time delay

-Hybrid Linear Systems – provide access to dynamic variables

-Variables that evolve continuously (such as via a differential equation).

-Altitude, time, speed, temperature….

-Very tricky to model and model check (HyTECH system can do it on occasion)

-Timed Temporal Logic (TCTL)

-Used to state properties about timed automata

-Extension of CTL

-Extends U,F,… operators with info on the flow time

-Ex: $pU_{<2}q$ means that p is true until q, where q is true in less than 2 time units from the current time

-TCTL syntax:

-$\phi_1, \phi_2 := p \mid ! \phi_1 \mid (\phi_1 \text{ and } \phi_2) \mid (\phi_1 \rightarrow \phi_2) \mid (\phi_1 \text{ or } \phi_2) \mid EF_{(\sim k)} \phi_1 \mid EG_{(\sim k)} \phi \mid E[\phi_1 U_{(\sim k)} \phi_2] \mid AF_{(\sim k)} \phi_1 \mid AG_{(\sim k)} \phi_1 \mid A[\phi_1 U_{(\sim k)} \phi_2]$

-Where ~ is any comparison (<, >, =, …)

-We'll see some examples of formulas in class

-Note: X operator doesn't exist because clocks have real values, so there is no notion of "next configuration"

-So how do we performed Timed Model Checking?

-Problem: infinite number of configurations because clocks take on real values => infinitely many valuations

-How fix?

-Define a notion of "closeness" between configurations

-Given clock constraints appearing in transitions and largest constraint used in these constraints, equivalence (~) on clock valuations is defined with the following property: for any timed automaton using these constraints, 2 configurations (q,v) and (q,v') with v ~ v' satisfy the same TCTL formulas

-This defines a set of equivalence classes (or regions). There is a finite number of regions!

-Given a configuration (q,v), we consider instead the region [v] for v.

-This defines a global automaton, or a region graph that represents abstractly the system. We model check on that instead

-Configurations are grouped into a region depending on their clock valuations

-One problem: exponential in number of clocks

- Timed Automata are relatively new, but some progress is still being made
- We'll see a full example of a region graph in class

- Time permitting, we will discuss some more about SMV (via a full example) to prepare you for PS4