CSEP590 – Model Checking and Automated Verification

Lecture outline for July 30, 2003

-We will first finish up Timed Automata from the last lecture…

-The fixed point characterization of CTL

- -We discuss this issue to motivate a proof of correctness of our model checking algorithm for CTL
- -This also provides necessary background for discussing the relational mu-calculus and its applications to model checking

-Recall: given a Model M = (S,→,L), our algorithm computes all $s \in S$ s.t. $M,s \models \phi$ for a CTL formula ϕ

- -We denote this set as $\{\phi\}$
- -Our algorithm is recursive on the structure of ϕ
- -For boolean operators it is easy to find $\{\phi\}$ via combinations of subsets using Union, Intersection, etc
- -An interesting case though is a formula involving a temporal operator (such as EX ϕ)
  - -We compute the set $\{\phi\}$, then compute the set of all states with transitions to a state in $\{\phi\}$
- -How do we reason about EU,AF, and EG? – we are iterating a labelling policy until stabilised!

-But how do we know that such iterations will terminate and even return the correct sets?? How can we argue this?

-Defn: let S be a set of states and F: P(S) → P(S) be a function on the power set of S (where P(S) denotes power set of S). Then,

- -1) F is monotone if $x \subseteq Y$ implies that $F(X) \subseteq F(Y)$ for all subsets X and Y of S
- -2) A subset X of S is called a fixed point of F if F(X) = X

-We'll see an example in class of fixed points and monotone functions. Indeed, a greatest fixed point is a subset X that is a fixed point and has the largest size. A least fixed point can be defined similarly

-Why are we exploring monotone functions?

- -They always have a least and greatest fixed point
- -The meanings of EG,AF,EU can be expressed via greatest and least fixed points of monotone function on P(S) (S = set of states)
- -Fixed points are easily computed

-Notation: $F^i(X) = F(F(\ldots F(X)\ldots))$ => a function F applied i times

-Theorem: Let S be a set $\{s_0, s_1, \ldots s_n\}$ with n+1 elements. If $F: P(S) \rightarrow P(S)$ is a monotone function, then $F^{n+1}(\emptyset)$ is the least fixed point of F, and $F^{n+1}(S)$ is the greatest fixed point of F.

- -Proof: in book on page 207
- -This theorem provides a recipe for computing fixed points! Indeed, the method is bounded at n+1 iterations.

-Now, we can prove the correctness of our model checking algorithm

- -Proof that EG algorithm is correct:
  - -We could say that $EG\ \phi = \phi \wedge EXEG\ \phi$ (call this (1))
  - -Also, $\{EG\ \phi\} = \{s \mid \text{exists } s' \text{ s.t. } s \rightarrow s' \text{ and } s' \in \{\phi\}\}$
  - -Thus, we can rewrite (1) as
    - -$\{EG\ \phi\} = \{\phi\} \cap \{s \mid \text{exists } s' \text{ s.t. } s \rightarrow s' \text{ and } s' \in \{EG\ \phi\}\}$
    - -Thus, we calculate $\{EG\ \phi\}$ from $\{EG\ \phi\}$ – this sounds like a fixed point operation!
  - -Indeed, $\{EG\ \phi\}$ is a fixed point of the function
    - -$F(X) = \{\phi\} \cap \{s \mid \text{exists } s' \text{ s.t. } s \rightarrow s' \text{ and } s' \in X\}$

- F is monotone, and {EG ϕ} is its greatest fixed point
    - (Formal proof is in book on pg. 209)
- {EG ϕ} can be computed using our theorem for fixed points, applied iteratively
    - ie, {EG ϕ} = $F^{n+1}(S)$ where $n+1=|S|$
- Thus, correctness of EG procedure is proved and it is guaranteed to terminate in at most $|S|$ iterations

-The book gives similar fixed point analysis for the EU operator, showing that its algorithm is also correct

-This, when combined with the correctness of EX and the boolean operators, completes proof of correctness of our CTL model checking algorithm

-Now, let's discuss the relational mu-calculus and how model checking can be performed in it

-We introduce a syntax for referring to fixed points in the context of boolean formulas

-Formulas of the relational mu-calculus grammar:

- -t = x | Z
- -f = 0 | 1| t | !f | $f_1 + f_2$ | $f_1 * f_2$ | ∃x.f | ∀x.f | uZ.f | vZ.f | f[X=X']
- -Where x is a boolean variable, Z is a relational variable, and X is a tuple of variables
- -A relational variable can be assigned a subset of S (set of states)
- -In formulas uZ.f and vZ.f any occurrence of Z in f is required to fall within an even # of complementation symbols
  - -Such an f is called formally monotone in Z
- -Symbols u and v stand for least and greatest fixed point operators
  - -Thus, uZ.f means "least fixed point of function f" (where the iteration is "occuring" on relational variable Z. The "returned" Z is the least fixed point of f)

- The formula f[X=X'] expresses the explicit substitution forcing f to be evaluated using the values of x_i' rather than x_i (allows for notions of "next time" evaluations, like successors)

-A valuation p for f is an assignment of values 0 or 1 to all variables

-Define: satisfaction relation p |= f inductively over the structure of such formulas f, given a valuation p

-We define |= for formulas without fixed point operators:

- -p !|= 0, p |= 1, p |= v iff p(v)=1, p |= !f iff p !|= f, p |= f+g iff p |= f or p |= g, p |= f*g iff p |= f and p |= g, p |= ∃x.f iff p[x=0] |= f or p[x=1] |= f, p |= ∀x.f iff p[x=0] |= f and p[x=1] |= f, p |= f[X=X'] iff p[X=X'] |= f
- -Where p[X=X'] is the valuation assigning the same values as p but for each x_i in X, it assigns p(x_i')
- -We'll see a few examples in class that make all this jumbled notation clearer

-Now, we extend the |= definition to fixed point operators u and v

-p |= uZ.f iff p |= u_mZ.f for some m >= 0

-Where uZ.f is recursively defined as

-u_0Z.f = 0

-u_mZ.f = f[u_{m-1}Z.f/Z] (that is, replace all occurrences of Z in f with u_{m-1}Z.f)

-p |= vZ.f iff p |= v_mZ.f for *all* m >= 0

-Where vZ.f is recursively defined as

-v_oZ.f = 1

-v_mZ.f = f[v_{m-1}Z.f/Z]

-We'll see some examples in class that will makes this intuitive. Essentially, these are just recursive definitions, they iterate to fixed points

-So now we can code CTL models and specifications

-Given a model M=(S,→,L), the u and v operators permit us to translate any CTL formula ϕ into a formula f^{ϕ} of the relational mu-calculus s.t. f^{ϕ} represents the set of states s where s |= ϕ

-Then, given a valuation p (ie, a state), we can check if p |= f^{ϕ}, meaning that the state satisfies ϕ

-Indeed, we can do this purely symbolically

- -Recall that the transition relation $\rightarrow$ can be represented as a boolean formula $f^{\rightarrow}$ (from our symbolic model checking lecture 4). Also, sets of states can be encoded as boolean formulas

-Therefore, the coding of a CTL formula ϕ as a function f^{ϕ} in relational mu-calculus is given inductively:

- $-f^{x} = x$ for vars x
- $-f^{\perp} = 0$
- $-f^{!\phi} = !f^{\phi}$
- $-f^{\phi \vee \varphi} = f^{\phi} * f^{\varphi}$
- $-f^{EX\phi} = \exists X'.(f^{\rightarrow} * f^{\phi}[X=X'])$
    - -What the heck does that mean? "There exists a next state s.t. the transition relation holds from the current state AND f^{ϕ} holds in this next state"

-We can also encode the formula for EFϕ

-Note that $EF\phi = \phi \vee EXEF\phi$

-Thus, $f^{EF\phi}$ is equivalent to f^{ϕ} + $f^{EXEF\phi}$, which is equivalent to $f^{\phi} + \exists X'.(f^{\rightarrow} * f^{EF\phi}[X=X'])$

-Since EF involves computing the least fixed point, we obtain

- $-f^{EF\phi} = uZ.(f^{\phi} + \exists X'.(f^{\rightarrow} * Z[X=X']))$, where Z is a relational variable.
- -Thus, we are getting the least fixed point of the formula that precisely encodes $EF\phi = \phi \vee EXEF\phi$

-The book provides similar coding for AF and EG on page 368

-The important point is to see how we used the fixed point characterization of CTL to code CTL formulas in relational mu-calculus (which has a fixed point syntax!)

-Thus, we can model check in terms of these relational mu-calculus formulas and symbolic representations of states and the transition relation

-Our last topic today, time-permitting, is to discuss a few abstraction techniques in model checking

- -Abstraction methods are a family of techniques used to simplify automata.
    - -It is probably "the most important technique for reducing the state explosion problem." –EM Clarke
- -Aim: given model as an automata A, we reduce a complex problem of $A \models \phi$ into a much simpler problem $A' \models \phi$
    - -Thus, this is another layer of abstraction on top of the abstraction of specifying a model to represent the system in question
- -We'll look more at examples to illustrate abstraction as opposed to developing a formal theory (for those interested, see me after class or email)
- -Why/when abstraction? Automata (model) is too big to check, of model checker doesn't handle certain details of the model

- We'll look at 2 techniques
  - Abstraction by state merging
  - Cone of influence reduction
- Abstraction by state merging
  - View some states as identical (ie, notions of folding states)
  - Merged states are put together into a super-state
  - Merging can be used for verifying safety properties, mainly because
    - 1) the merged automata A' has more behaviors than A
    - 2) the more behaviors an automata has, the fewer safety properties it fulfills
    - 3) thus, if A' satisfies a safety property p, then so too does A satisfy p
    - 4) if A' doesn't satisfy p, *no* conclusion can be drawn about A
      - Why is this verification only one-way?
  - There is a difficulty here though:

- How are atomic propositions labeling states gathered together into the super-state??
  - In principle: never merge states that are labeled with different sets of atomic props
  - But this is way too restrictive
  - How weaken?
    - Turns out that if merging is used to check property p, then only the propositions occurring in p are relevant
    - Thus, if a proposition X only appears in positive form in p (each occurrence of X is within an even # of negation symbols), then we can merge states w/o the need for these to agree on the presence of X
    - The super-state then carries the label of X iff all merged states carry the label X
    - This rationale isn't obvious though…

- Abstraction via cone of influence reduction
  - Suppose we are given a subset of the variables $V' \subseteq V$ that are of interest with respect to a required spec

-Recall: system can be specified as a Kripke Structure using equations for transition relations, and an equation for the initial set of states of the system

-We want to simplify the system description by referring to only those variables V'

-But, values of V' variables may depend on the values of variables not in V'

  -For example, we'll consider the modulo 8 counter that we examined in lecture 2

-We define the cone of influence C for V' and use C for our reduction of the system

-Defn: the cone of influence C of V' is the minimal set of vars s.t.

  -1) V' is a subset of C

  -2) if for some $v_l \in C$ its formula f_l depends on v_j, then v_j is also in C

-Therefore, the reduced system is constructed by removing all transition equations whose left hand side variables do not appear in C

-We’ll see the full example of this technique in class using the Kripke Structure model for the modulo 8 counter

-We won’t, however, go over the proof arguing that removal of such equations doesn’t affect the equivalency of the model