Automata-Based Model Checking

Yih-Kuen Tsay

Dept. of Information Management
National Taiwan University

Outline

Model Checking

- The Problem
  Determining if the specification is true of a (finite-state concurrent) system, i.e., *checking* if the system is a *model* of the specification
- The Process
  - Modeling: convert a design into a formal model
    Main systems considered: *finite-state transition systems* (modeling digital circuits, communication protocols, etc.)
  - Specification: state the properties that the design must satify
    Typical specification languages: *propositional modal/temporal logics*
  - Verification: is automatic ideally, but may involve human assistance in practice

Model Checking (cont.)

- Advantages (over deductive verification methods):
  - Fully automatic
  - Providing counterexamples
- Main obstacle: the state explosion problem
- Became practically viable with symbolic encoding
- Has been most successful in verifying hardware and communication protocols
- Commercial model checking tools in the market

Formal Modeling

- First two steps in correctness verification:
  1. Specify the desired *properties*
  2. Construct a *formal model* (with the desired properties in mind)
     - Capture the necessary properties and leave out the irrelevant
     - Example: gates and boolean values vs. voltage levels
     - Example: exchange of messages vs. contents of messages
- Description of a formal model
  - Graphs
  - Logic formulae

Concurrent Reactive Systems

- A typical type of systems that model checking techniques deal with
- Interact frequently with the environment and *may not terminate*
- *Temporal* (not just input-output) behaviors are most important
- Modeling elements:
  - State: a snapshot of the system at a particular instance
  - Transition:
    - how the system changes its state as a result of some action
    - described by a pair of the state before and the state after the action
  - Computation: an infinite sequence of states resulted from transitions

Kripke Structures

IM NTU

- Kripke structures are one of the most popular types of formal models for concurrent systems.
- Let AP be a set of atomic propositions (representing things you want to observe).
- A *Kripke structure* M over AP is a tuple $\langle S, S_0, R, L \rangle$:
  - S is a finite set of states,
  - $S_0 \subseteq S$ is the set of initial states,
  - $R \subseteq S \times S$ is a *total* transition relation, and
  - $L : S \rightarrow 2^{AP}$ is a function labeling each state with a subset of propositions (which are true in that state).
- A *computation* or *path* of M from a state s is an infinite sequence of states $\sigma = s_0, s_1, s_2, \cdots$ such that $s_0 = s$ and $(s_i, s_{i+1}) \in R$, for all $i \geq 0$.

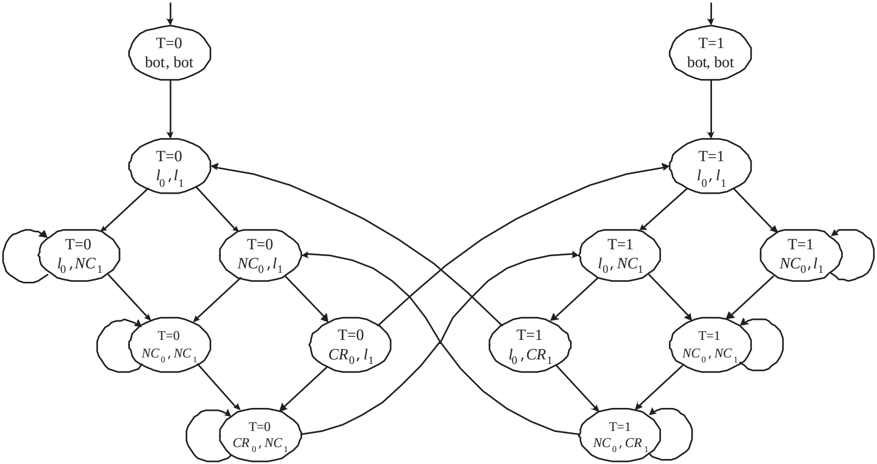
Example: Mutual Exclusion Program P_{MX}

$$P_{MX} = m : \textbf{cobegin } P_0 \parallel P_1 \textbf{ coend } m'$$

$P_0 =$
$l_0 :$ **while** *True* **do**
  $NC_0 :$ **wait** $T = 0;$
  $CR_0 : T := 1;$
od;
l'_0

$P_1 =$
$l_1 :$ **while** *True* **do**
  $NC_1 :$ **wait** $T = 1;$
  $CR_1 : T := 0;$
od;
l'_1

A Kripke Structure for P_{MX}



Source: redrawn from [Clarke et al. 1999, Fig 2.2]

Properties

- About the computations of a system and typically temporal
- Types of properties
  - Safety: "something bad" does not happen
  - Liveness: "something good" will eventually happen
- Examples
  - Two processes are never in the critical section at the same time. (safety)
  - A request always gets a reply. (liveness)
- Two commonly used specification formalisms
  - Temporal logic (linear-time vs. branching-time)
  - Automata (on infinite objects)

Outline

Automata for Modeling Infinite Behaviors

- The simplest computation model for finite behaviors is the finite state automaton, which accepts finite words.
- The simplest computation model for infinite behaviors is the ω-automaton, which accepts infinite words.
- Both have the same syntactic structure.
- Model checking traditionally deals with non-terminating systems.
- Infinite words conveniently represent the infinite behaviors exhibited by a non-terminating system.
- Büchi automata are the simplest kind of ω-automata.
- They were first proposed and studied by J.R. Büchi in the early 1960's, to devise decision procedures for S1S (a second-order theory).

Büchi Automata

- A Büchi automaton (BA) has the same structure as a finite state automaton (FA) and is also given by a 5-tuple $(\Sigma, Q, \Delta, q_0, F)$:
  1. Σ is a finite set of symbols (the *alphabet*),
  2. Q is a finite set of *states*,
  3. $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*,
  4. $q_0 \in Q$ is the *start* state (sometimes we allow multiple start states, indicated by Q_0 or Q^0), and
  5. $F \subseteq Q$ is the set of *accepting* (final in FA) states.
- Let $B = (\Sigma, Q, \Delta, q_0, F)$ be a BA and $w = w_1 w_2 \ldots w_i w_{i+1} \ldots$ be an infinite string (or word) over Σ.
- A *run* of B over w is a sequence of states $r_0, r_1, r_2 \ldots, r_i r_{i+1} \ldots$ such that
  1. $r_0 = q_0$ and
  2. $(r_i, w_{i+1}, r_{i+1}) \in \Delta$ for $i \geq 0$.

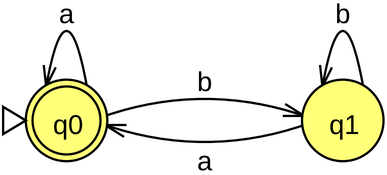
Büchi Automata (cont.)

- Let $\mathit{inf}(\rho)$ denote the set of states occurring infinitely many times in a run ρ.
- An infinite word $w \in \Sigma^\omega$ is *accepted* by a BA B if there exists a run ρ of B over w satisfying the condition:

$$\mathit{inf}(\rho) \cap F \neq \emptyset.$$

- The *language* recognized by B (or the language of B), denoted $L(B)$, is the set of all words that are accepted by B.

An Example Büchi Automaton



- This Büchi automaton accepts infinite words over $\{a, b\}$ that have infinitely many a's.
- Using an ω-regular expression, its language is expressed as $(b^* a)^\omega$.

Closure Properties

- A class of languages is closed under intersection if the intersection of any two languages in the class remains in the class.
- Analogously, for closure under complementation.

Theorem

The class of languages recognizable by Büchi automata is closed under **intersection** *and* **complementation** *(and hence all boolean operations).*

Proof.

Closure under intersection will be proven later by giving a procedure for constructing a Büchi automaton that recognizes the intersection of the languages of two given Büchi automata.
Closure under complementation will be proven in a separate lecture. □

Generalized Büchi Automata

IM NTU

- A generalized Büchi automaton (GBA) has an acceptance component of the form $F = \{F_1, F_2, \cdots, F_n\} \subseteq 2^Q$.
- A run ρ of a GBA is accepting if for each $F_i \in F$, $\mathit{inf}(\rho) \cap F_i \neq \emptyset$.
- GBA's naturally arise in the modeling of finite-state concurrent systems with fairness constraints.
- They are also a convenient intermediate representation in the translation from a linear temporal formula to an equivalent BA.
- There is a simple translation from a GBA to a Büchi automaton, as shown next.

GBA to BA

- Let $B = (\Sigma, Q, \Delta, Q^0, F)$, where $F = \{F_1, \cdots, F_n\}$, be a GBA.
- Construct $B' = (\Sigma, Q \times \{0, \cdots, n\}, \Delta', Q^0 \times \{0\}, Q \times \{n\})$.
- The transition relation Δ' is constructed such that $(\langle q, x\rangle, a, \langle q', y\rangle) \in \Delta'$ when $(q, a, q') \in \Delta$ and x and y are defined according to the following rules:
  - If $q' \in F_i$ and $x = i - 1$, then $y = i$.
  - If $x = n$, then $y = 0$.
  - Otherwise, $y = x$.
- Claim: $L(B') = L(B)$.

Theorem

For every GBA B, there is an equivalent BA B' such that $L(B') = L(B)$.

Outline

Model Checking Using Automata

- Finite automata can be used to model concurrent and reactive systems as well.
- One of the main advantages of using automata for model checking is that both the modeled system and the specification are represented in the same way.
- A Kripke structure directly corresponds to a Büchi automaton, where all the states are accepting.
- A Kripke structure (S, R, S_0, L) can be transformed into an automaton $A = (\Sigma, S \cup \{\iota\}, \Delta, \{\iota\}, S \cup \{\iota\})$ with $\Sigma = 2^{AP}$ where
  - $(s, \alpha, s') \in \Delta$ for $s, s' \in S$ iff $(s, s') \in R$ and $\alpha = L(s')$ and
  - $(\iota, \alpha, s) \in \Delta$ iff $s \in S_0$ and $\alpha = L(s)$.

Model Checking Using Automata (cont.)

- The given system is modeled as a Büchi automaton A.
- Suppose the desired property is originally given by a linear temporal formula f.
- Let B_f (resp. $B_{\neg f}$) denote a Büchi automaton equivalent to f (resp. $\neg f$); we will later study how a temporal formula can be translated into an automaton.
- The model checking problem $A \models f$ is equivalent to asking whether

$$L(A) \subseteq L(B_f) \text{ or } L(A) \cap L(B_{\neg f}) = \emptyset.$$

- The well-used model checker SPIN, for example, adopts this automata-theoretic approach.
- So, we are left with two basic problems:
  - Compute the intersection of two Büchi automata.
  - Test the emptiness of the resulting automaton.

Outline

Intersection of Büchi Automata

- Let $B_1 = (\Sigma, Q_1, \Delta_1, Q_1^0, F_1)$ and $B_2 = (\Sigma, Q_2, \Delta_2, Q_2^0, F_2)$.
- We can build an automaton for $L(B_1) \cap L(B_2)$ as follows.
- $B_1 \cap B_2 =$
  $(\Sigma, Q_1 \times Q_2 \times \{0, 1, 2\}, \Delta, Q_1^0 \times Q_2^0 \times \{0\}, Q_1 \times Q_2 \times \{2\})$.
- We have $(\langle r, q, x \rangle, a, \langle r', q', y \rangle) \in \Delta$ iff the following conditions hold:
  - $(r, a, r') \in \Delta_1$ and $(q, a, q') \in \Delta_2$.
  - The third component is affected by the accepting conditions of B_1 and B_2.
    - If $x = 0$ and $r' \in F_1$, then $y = 1$.
    - If $x = 1$ and $q' \in F_2$, then $y = 2$.
    - If $x = 2$, then $y = 0$.
    - Otherwise, $y = x$.
- The third component is responsible for guaranteeing that accepting states from both B_1 and B_2 appear infinitely often.

Intersection of Büchi Automata (cont.)

- A simpler intersection may be obtained when all of the states of one of the automata are accepting.
- Assuming all states of B_1 are accepting and that the acceptance set of B_2 is F_2, their intersection can be defined as follows:

$$B_1 \cap B_2 = (\Sigma, Q_1 \times Q_2, \Delta', Q_1^0 \times Q_2^0, Q_1 \times F_2)$$

where $(\langle r, q \rangle, a, \langle r', q' \rangle) \in \Delta'$ iff $(r, a, r') \in \Delta_1$ and $(q, a, q') \in \Delta_2$.

Checking Emptiness

- Let ρ be an accepting run of a Büchi automaton $B = (\Sigma, Q, \Delta, Q^0, F)$.
- Then, ρ contains infinitely many accepting states from F.
- Since Q is finite, there is some suffix ρ' of ρ such that every state on it appears infinitely many times.
- Each state on ρ' is reachable from any other state on ρ'.
- Hence, the states in ρ' are included in a strongly connected component.
- This component is reachable from an initial state and contains an accepting state.

Checking Emptiness (cont.)

- Conversely, any strongly connected component that is reachable from an initial state and contains an accepting state generates an accepting run of the automaton.
- Thus, checking nonemptiness of $L(B)$ is equivalent to finding a strongly connected component that is reachable from an initial state and contains an accepting state.
- That is, the language $L(B)$ is nonempty iff there is a reachable accepting state with a cycle back to itself.

Double DFS Algorithm

IM NTU

```
procedure emptiness
    for all q₀ ∈ Q⁰ do
        dfs1(q₀);
    terminate(True);
end procedure

procedure dfs1(q)
    local q′;
    hash(q);
    for all successors q′ of q do
        if q′ not in the hash table then dfs1(q′);
    if accept(q) then dfs2(q);
end procedure
```

Double DFS Algorithm (cont.)

```
procedure dfs2(q)
    local q';
    flag(q);
    for all successors q' of q do
        if q' on dfs1 stack then terminate(False);
        else if q' not flagged then dfs2(q');
        end if;
end procedure
```

Correctness

Lemma

Let q be a node that does not appear on any cycle. Then the DFS algorithm will backtrack from q only after all the nodes that are reachable from q have been explored and backtracked from.

Theorem

The double DFS algorithm returns a counterexample for the emptiness of the checked automaton B exactly when the language $L(B)$ is not empty.

Correctness (cont.)

- Suppose a second DFS is started from a state q and there is a path from q to some state p on the search stack of the first DFS.
- There are two cases:
  - There exists a path from q to a state on the search stack of the first DFS that contains only unflagged nodes when the second DFS is started from q.
  - On every path from q to a state on the search stack of the first DFS there exists a state r that is already flagged.
- The algorithm will find a cycle in the first case.
- We show that the second case is impossible.

Correctness (cont.)

- Suppose the contrary: On every path from q to a state on the search stack of the first DFS there exists a state r that is already flagged.
- Then there is an accepting state from which a second DFS starts but fails to find a cycle even though one exists.
  - Let q be the first such state.
  - Let r be the first flagged state that is reached from q during the second DFS and is on a cycle through q.
  - Let q' be the accepting state that starts the second DFS in which r was first encountered.
- Thus, according to our assumptions, a second DFS was started from q' before a second DFS was started from q.

Correctness (cont.)

- Case 1: The state q' is reachable from q.
  - There is a cycle $q' \to \cdots \to r \to \cdots \to q \to \cdots \to q'$.
  - This cycle could not have been found previously.
  - This contradicts our assumption that q is the first accepting state from which the second DFS missed a cycle.
- Case 2: The state q' is not reachable from q.
  - q' cannot appear on a cycle.
  - q is reachable from r and q'.
  - If q' does not occur on a cycle, by Lemma 23 we must have backtracked from q in the first DFS before from q'.
  - This contradicts our assumption about the order of doing the second DFS.

Outline

Concluding Remarks

- Properties of a system are more conveniently specified by linear temporal logic formulae.
- In a separate lecture, we will study how a linear temporal logic formula can be translated into an equivalent Büchi automaton.

References

- J.R. Büchi. On a decision method in restricted second-order arithmetic, in *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, Stanford University Press, 1962.
- E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*, The MIT Press, 1999.
- E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games* (LNCS 2500), Springer, 2002.
- G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, 2003.
- W. Thomas. Automata on infinite objects, *Handbook of Theoretical Computer Science* (Vol. B), 1990.