An Automata Theoretic Approach to Branching Time Model Checking

Arie Gurfinkel
arie@cs.toronto.edu
University of Toronto

CTL Satisfiability

- Bounded branching property of CTL
  - A CTL formula $\varphi$ is satisfiable iff it is satisfiable on a tree with branching degree bounded by $|\varphi|$
- Satisfiability via alternating automata
  - Given a CTL formula $\varphi$
    - build an ATA $A_\varphi$ that works over $|\varphi|$-ary trees
    - $\varphi$ is satisfiable iff $A_\varphi$ is non-empty
- Non-emptiness problem for ATA is EXPTIME-complete
  - Satisfiability problem for CTL is in EXPTIME
CTL Model-Checking

- $K$ is a Kripke structure with branching degrees in $\mathcal{D}$
- $\varphi$ is a CTL formula
- Build an ATA $A_{\mathcal{D},\varphi}$ such that
  - $\mathcal{L}(A_{\mathcal{D},\varphi})$ contains all $\mathcal{D}$-trees that satisfy $\varphi$
- $K \models \varphi$ iff the computation tree $T_K$ induced by $K$ is in $\mathcal{L}(A_{\mathcal{D},\varphi})$
- Automata-based model-checking algorithm
  - build a product automaton of $K$ and $A_{\mathcal{D},\varphi}$ whose language is $\mathcal{L}(A_{\mathcal{D},\varphi}) \cap T_K$
  - check if it is empty or not

Complexity

- In general, checking non-emptiness of an ATA is expensive
- But, we are dealing with a special case
  - only interested in automata that arise from CTL formulas
  - the language of the product automaton either empty, or contains a single tree
- Using the above we obtain a linear automata-based CTL algorithm
Weak Alternating Automata

A Büchi ATA $A = (\Sigma, Q, q_0, \delta, F)$ is weak iff

- exists a partitioning of $Q$ into $Q_1, Q_2, \ldots, Q_n$
- each $Q_i$ is either accepting
  - $Q_i \subseteq F$
- or rejecting
  - $Q_i \cap F = \emptyset$
- there exists a partial order such that
  - if $q \in Q_i$, $q_j \in Q_j$, and $q_j$ appears in the transition of $q$, then $i \geq j$

Any run of a Weak Alternating Automaton (WAA) gets trapped in either accepting or rejecting state

Weak Alternating Automata and CTL

ATA constructed for a CTL formula $\varphi$ is weak

- each formula in $cl(\varphi)$ forms a singleton set in the partition
- the partial order is given by the sub-formula ordering

Let $\varphi = EF(AGp)$, then

- the partition is $\{EF(AGp)\} > \{AGp\} > \{p\}$
- $\{AGp\}$ is the accepting set
- $\{EF(AGp)\}$, and $\{p\}$ are rejecting
The Product Automaton

- The product automaton of ATA $A_\varphi$ and a Kripke structure $K$ is
  - weak if $A_\varphi$ is weak
  - Büchi if $A_\varphi$ is Büchi
  - an automaton over infinite words!
  - a 1-letter automaton!
    - i.e. $\Sigma = \{a\}$

- The product automaton is weak alternating Büchi word automaton

Constructing the Product

- Let $A_\varphi = (2^{AP}, Q, q_0, \delta_\varphi, F)$ be a WAA for CTL formula $\varphi$
- Let $K = (AP, S, s_0, L, R)$ be a Kripke structure
- The product automaton is $A_{K,\varphi} = (\{a\}, Q \times S, \langle q_0, s_0 \rangle, \delta, S \times F)$
  - if $\delta_\varphi(q, L(s), k) = \theta$
  - and $R(s) = t_0, \ldots, t_k$, then
    - $\delta(\langle q, s \rangle, a) = \theta'$,
    - where $\theta'$ is obtained from $\theta$ by replacing each $(c, q')$ with $\langle q', t_c \rangle$
- The weakness partition is induced by $A_\varphi$
- The product automaton simulates the run of $A_\varphi$ on the computation tree of $K$
### Example

#### Formula

\( \varphi = EFq \)

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, \emptyset, k) )</th>
<th>( \delta(q, {q}, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>( EFq )</td>
<td>( \bigvee_{c=0}^{k-1} (c, EFq) )</td>
<td>true</td>
</tr>
</tbody>
</table>

#### Kripke Structure

![Kripke Structure Diagram](image)

#### Product Automaton

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle EFq, s_0 \rangle )</td>
<td>( \langle EFq, s_1 \rangle )</td>
</tr>
</tbody>
</table>

### Example

#### Formula

\( \varphi = EGAXp \)

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, \emptyset, k) )</th>
<th>( \delta(q, {p}, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>( AXp )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) )</td>
</tr>
<tr>
<td>( EGAXp )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) \land \bigvee_{c=0}^{k-1} (c, EGAXp) )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) \land \bigvee_{c=0}^{k-1} (c, EGAXp) )</td>
</tr>
</tbody>
</table>

#### Product Automaton

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle p, s_0 \rangle )</td>
<td>true</td>
</tr>
<tr>
<td>( \langle p, s_1 \rangle )</td>
<td>true</td>
</tr>
<tr>
<td>( \langle EGAXp, s_0 \rangle )</td>
<td>( \langle p, s_1 \rangle \land \langle EGAXp, s_1 \rangle )</td>
</tr>
<tr>
<td>( \langle EGAXp, s_1 \rangle )</td>
<td>( \langle p, s_0 \rangle \land \langle p, s_1 \rangle \land (\langle EGAXp, s_0 \rangle \lor \langle EGAXp, s_1 \rangle) )</td>
</tr>
</tbody>
</table>
Non-Emptiness Algorithm

The algorithm proceeds up in the weakness partial order

- each state is labeled with either true or false
- an automaton is non-empty iff its initial state is labeled with true

The algorithm

- pick a \( Q_i \) with the smallest \( i \) that is not yet labeled
- repeatedly, for each \( q \in Q_i \)
  - label \( q \) with true is \( \delta(q, a) = \text{true} \)
  - label \( q \) with false is \( \delta(q, a) = \text{false} \)
  - use the labeling on \( q \) to simplify any \( \delta \) in which \( q \) occurs
- if there are any unlabeled states in \( Q_i \)
  - if \( Q_i \) is accepting, label them with true
  - if \( Q_i \) is rejecting, label them with false

Complexity

- linear in the size of property automaton and the Kripke structure
Example

\[ \varphi = EFq \]

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, \emptyset, k) )</th>
<th>( \delta(q, {q}, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>( EFq )</td>
<td>( \bigvee_{c=0}^{k-1} (c, EFq) )</td>
<td>true</td>
</tr>
</tbody>
</table>

Kripke Structure

Product automaton

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle EFq, s_0 \rangle )</td>
<td>( \langle EFq, s_1 \rangle )</td>
</tr>
<tr>
<td>( \langle EFq, s_1 \rangle )</td>
<td>true</td>
</tr>
</tbody>
</table>

Example

\[ \varphi = EGAXp \]

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, \emptyset, k) )</th>
<th>( \delta(q, {p}, k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>( AXp )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) )</td>
</tr>
<tr>
<td>( EGAXp )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) \land \bigvee_{c=0}^{k-1} (c, EGAXp) )</td>
<td>( \bigwedge_{c=0}^{k-1} (c, p) \land \bigvee_{c=0}^{k-1} (c, EGAXp) )</td>
</tr>
</tbody>
</table>

Product automaton

<table>
<thead>
<tr>
<th>state</th>
<th>( \delta(q, a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle p, s_0 \rangle )</td>
<td>true</td>
</tr>
<tr>
<td>( \langle p, s_1 \rangle )</td>
<td>true</td>
</tr>
<tr>
<td>( \langle EGAXp, s_0 \rangle )</td>
<td>( \langle p, s_1 \rangle \land \langle EGAXp, s_1 \rangle )</td>
</tr>
<tr>
<td>( \langle EGAXp, s_1 \rangle )</td>
<td>( \langle p, s_0 \rangle \land \langle p, s_1 \rangle \land (\langle EGAXp, s_0 \rangle \lor \langle EGAXp, s_1 \rangle) )</td>
</tr>
</tbody>
</table>
Space Efficient Algorithm

- For LTL we have an on-the-fly algorithm that only builds as much of the structure as necessary
- Is this possible for CTL?
- For a long time it was considered that the bottom-up nature of the CTL algorithm requires to construct the full structure first!
- With HAA we show that a space efficient algorithm for CTL is possible
- Actually, the same algorithm applies to CTL*  
- But, not to the alternation free $\mu$-calculus!

Hesitation Partition

$A_{K,\varphi} = K \times A_{D,\varphi}$ — product automaton of $K$ and $\varphi$
- each state is an element of $S \times cl(\varphi)$
- weakness partition based on the second component
- at most $|cl(\varphi)|$ elements of the partition
- Each partition set $Q_i$ can be classified as
  - transient — all transitions lead to states in lower $Q_i$’s
    - corresponds to all elements of $cl(\varphi)$ except for $U$ and $R$ formulas
  - existential — a transition only contains disjunctively related elements of the same $Q_i$
    - corresponds to $EU$ and $ER$ formulas
  - universal — a transition only contains conjunctively related elements of the same $Q_i$
    - corresponds to $AU$ and $AR$ formulas
Hesitant Automata

A = (Σ, D, Q, δ, q₀, (G, B)) is a hesitant automaton iff
- Q can be partitioned into sets Qᵢ
- each Qᵢ is either transient, existential, or universal
- there exists a partial order on the partition such that transitions in Qᵢ lead either to the same Qᵢ, or to a lower one
- G, B ⊆ Q is a acceptance condition
- the partial order is called hesitation order
- the longest chain in it is the hesitation depth

Each infinite path along a run of a HAA is trapped in either existential or universal Qᵢ
- an infinite path π is accepting if
  - Qᵢ is existential and Inf(π) ∩ G ≠ ∅
  - Qᵢ is universal and Inf(π) ∩ B = ∅

From CTL to HAA

The WAA constructed from a CTL formula already satisfies the hesitation partition
- we only need to change the acceptance condition

The acceptance condition of WAA contains all ER and AR sub-formulas
- a path on a run is allowed to get trapped only in a set corresponding to ER or AR formulas
From CTL to HAA

- In HAA we get
  - a path can get trapped in an existential set iff it corresponds to $ER$ formula
  - a path can get trapped in a universal set iff it does not correspond to $AU$ formula
- The acceptance condition for HAA is $\langle G, B \rangle$
  - $G$ contains all $ER$ sub-formulas
  - $B$ contains all $AU$ sub-formulas
- The transition relation of HAA is more constrained than of WAA, but its acceptance condition is more expressive

Example

$\varphi = EFq$

<table>
<thead>
<tr>
<th>state</th>
<th>$\delta(q, \emptyset, k)$</th>
<th>$\delta(q, {q}, k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>$EFq$</td>
<td>$\lor_{c=0}^{k-1}(c, EFq)$</td>
<td>true</td>
</tr>
</tbody>
</table>

acceptance condition ($\emptyset, \emptyset$)

$\varphi = EGAXp$

<table>
<thead>
<tr>
<th>state</th>
<th>$\delta(q, \emptyset, k)$</th>
<th>$\delta(q, {p}, k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>$AXp$</td>
<td>$\lor_{c=0}^{k-1}(c, p)$</td>
<td>$\lor_{c=0}^{k-1}(c, p)$</td>
</tr>
<tr>
<td>$EGAXp$</td>
<td>$\lor_{c=0}^{k-1}(c, p) \land \lor_{c=0}^{k-1}(c, EGAXp)$</td>
<td>$\lor_{c=0}^{k-1}(c, p) \land \lor_{c=0}^{k-1}(c, EGAXp)$</td>
</tr>
</tbody>
</table>

acceptance condition ($\{EGAXp\}, \emptyset$)
Non-Emptiness Algorithm for HAA

- Intuition
  - for a state from an existential set look for a witness
    - if one is found, label the state with true
    - otherwise, label it with false
  - for a state from a universal set look for a counterexample
    - if one is found, label the state with false
    - otherwise, label it with true
  - an automaton is non-empty iff the initial state is labeled with true
- Space complexity is $O(m \log^2 n)$
  - $m$ is the depth of the automaton
  - $n$ is its size

Immediate Reachability

- Let $q$ and $q'$ be states of the same $Q_i$
  - $q'$ is immediately reachable from $q$ iff
    - when $\delta(q, \sigma, k')$ is simplified using values of the states from the lower $Q_i$
    - $q'$ appears in $\delta$
    - i.e. the value of $\delta$ depends on $q'$
  - $q'$ is reachable from $q$ if there exists a path of immediate reachable states from $q$ to $q'$
  - A state $q$ is provably true if its transition simplifies to true
  - A state $q$ is provable false if its transition simplifies to false
Non-Emptiness Algorithm

- Start at the initial state
- If $q$ is a transient state recurse to all successors and simplify the transition relation
- If $q$ is from an existential $Q_i$
  - if there exists reachable state $q'$ in the same $Q_i$ that is provably true
    - label $q$ with true
  - if not, search for a reachable state $q' \in G$ in the same $Q_i$ that is reachable from itself
    - if found, label $q$ with true
    - otherwise, label $q$ with false
- If $q$ is from a universal $Q_i$
  - if there exists reachable state $q'$ in the same $Q_i$ that is provably false
    - label $q$ with false
  - if not, search for a reachable state $q' \in B$ in the same $Q_i$ that is reachable from itself
    - if found, label $q$ with false
    - otherwise, label $q$ with true
Example

φ = EFq

<table>
<thead>
<tr>
<th>state</th>
<th>δ(q, ∅, k)</th>
<th>δ(q, {q}, k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>EFq</td>
<td>$\bigvee_{c=0}^{k-1}(c, EFq)$</td>
<td>true</td>
</tr>
</tbody>
</table>

Kripke Structure

\[ \begin{array}{ccc}
\text{p} & \rightarrow & \text{q} \\
\text{s}_0 & \leftrightarrow & \text{q} \\
\text{q} & \rightarrow & \text{s}_1 \\
\end{array} \]

Product automaton

<table>
<thead>
<tr>
<th>state</th>
<th>δ(q, a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle EFq, s_0 \rangle</td>
<td>\langle EFq, s_1 \rangle</td>
</tr>
<tr>
<td>\langle EFq, s_1 \rangle</td>
<td>true</td>
</tr>
</tbody>
</table>

Example

φ = EGAXp

<table>
<thead>
<tr>
<th>state</th>
<th>δ(q, ∅, k)</th>
<th>δ(q, {p}, k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>AXp</td>
<td>$\bigwedge_{c=0}^{k-1}(c, p)$</td>
<td>$\bigwedge_{c=0}^{k-1}(c, p)$</td>
</tr>
<tr>
<td>EGAXp</td>
<td>$\bigwedge_{c=0}^{k-1}(c, p) \land \bigvee_{c=0}^{k-1}(c, EGAXp)$</td>
<td>$\bigwedge_{c=0}^{k-1}(c, p) \land \bigvee_{c=0}^{k-1}(c, EGAXp)$</td>
</tr>
</tbody>
</table>

Product automaton

<table>
<thead>
<tr>
<th>state</th>
<th>δ(q, a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle p, s_0 \rangle</td>
<td>true</td>
</tr>
<tr>
<td>\langle p, s_1 \rangle</td>
<td>true</td>
</tr>
<tr>
<td>\langle EGAXp, s_0 \rangle</td>
<td>\langle p, s_1 \rangle \land \langle EGAXp, s_1 \rangle</td>
</tr>
<tr>
<td>\langle EGAXp, s_1 \rangle</td>
<td>\langle p, s_0 \rangle \land \langle p, s_1 \rangle \land (\langle EGAXp, s_0 \rangle \lor \langle EGAXp, s_1 \rangle)</td>
</tr>
</tbody>
</table>
Summary

- Automata over infinite objects
  - many possible acceptance conditions
  - more expressive acceptance conditions lead to simpler automata for the same language
- Alternating automata
  - extend non-determinism by allowing both disjunctive and conjunctive choice
  - greatly simplify constructing property automata

Summary – Model-Checking

- automata provide a uniform solution to the model-checking problem
- branching versus linear time is captured by
  - automata over strings, and
  - automata over trees
- same solution to both satisfiability and model-checking
- a formula $\varphi$ is satisfiable iff
  - an automaton corresponding to $\varphi$ is non-empty
- a model $K$ satisfies a formula $\varphi$ iff
  - the product automaton of $K$ and $\varphi$ is non-empty
Summary – Model-Checking

- clean separation between logic and algorithms
- what does the formula mean?
- how to construct an automaton for it
- what is the complexity of model-checking
- solving the non-emptiness problem