Synthesis under incomplete information

Andreas Augustin

June 12, 2008
1 Outline
   - Background
   - Incomplete information

2 Automata types
   - Word automata
   - Alternating automata
   - Tree Automata
   - Alternating tree automata

3 Incomplete information
   - Overview
   - hide, wide and xray functions
   - Putting it all together
   - Final statements
We know automata that read input and make transitions
- finite
- infinite

You probably heard of automata that read input, produce output and make transitions (e.g. Moore, Mealy)

Behaviour of a reactive system

Program $P$ maps inputs $I$ and history to outputs $O$:

$$P : (2^I)^* \rightarrow 2^O$$
Specification and synthesis

- Specification as formula $\varphi$ in LTL, CTL, CTL*, $\mu$-calculus
- Realizability: Does there exist a program $P$ that satisfies $\varphi$?
- Synthesis: Transform specification $\varphi$ in program $P$ that is guaranteed to satisfy $\varphi$
Synthesis for LTL

- Specification yields allowed combinations of sequences of inputs and outputs
- Problem can be reduced to non-emptiness test of tree-automaton
- Synthesis is proven to be 2EXPTIME complete in this case
Synthesis for branching-time logics

- $P$ associates with each input sequence infinite computation over $2^{I \cup O}$
- $I$ and $O$ are disjoint, so $2^{I \cup O} = 2^I \times 2^O$
- Although $P$ deterministic, $P$ induces a computation tree due to external nondeterminism caused by different possible inputs in $I$
- Branching temporal logics (CTL, CTL*) give us the required expressive power because of path quantifiers: In LTL we can’t express possibility requirements.
- Realizability correlates to non-emptiness-test for tree-atomaton
Now assume the environment knows more than the program $P$:
- Signals $I$ of readable input
- Signals $E$ that are known to the environment, but unknown to $P$
- Signals $O$ as before

What’s the impact of this on
- Realizability?
- Complexity?
Example

- An adapted example from the paper[1]: Assume a printer scheduler shall only print a paper if it doesn’t contain bugs. Unfortunately, it can’t decide whether the paper contains a bug.

- We have:
  - \( I = \{i\}; \ i = 1 \iff \text{User wants to print a paper} \)
  - \( E = \{e\}; \ e = 1 \iff \text{Paper is buggy} \)
  - \( O = \{o\}; \ o = 1 \iff \text{Paper scheduled for printing} \)

- We want \( A\square(o \Rightarrow i \land \neg e) \)

- Since we can’t distinguish between \( i \land \neg e \) and \( i \land e \), the only safe way to handle this is never to print anything at all.
Word- and Tree-Automata and their alternating versions

- Word Automata
- Tree Automata
- Alternating Word A.
- Alternating Tree Automata
Word automata

- **Well known**
  - Alphabet $\Sigma$
  - States $Q$
  - Initial state(s) $i_0 \in Q$ or $I \subseteq Q$
  - Transition-relation or -function $\delta$, details follow
  - Acceptance condition $c$

- $\delta$ may vary depending on the type of automaton, determinism a.s.f.

- $c$ may be something like Muller-Acceptance, Rabin-Acceptance a.s.f.
Word Automata

A word automaton can be...

- **Deterministic.** Then $\delta$ is a function $\delta : Q \times \Sigma \rightarrow Q$
- **Nondeterministic.** Then $\delta$ is a relation $\delta : Q \times \Sigma \rightarrow 2^Q$
  - Instead of writing $\delta(q_1, \sigma) = \{q_2, q_3\}$ we can write $\delta(q_1, \sigma) = q_2 \lor q_3$ in the sense that the automaton accepts if proceeding in $q_2$ or $q_3$ accepts
- **Universal.** Then again, $\delta$ is a relation $\delta : Q \times \Sigma \rightarrow 2^Q$, but the automaton forks for each additional successor and we demand that all automatons accept
  - Again, we can write $\delta(q_1, \sigma) = q_2 \land q_3$, because the automaton that goes on in $q_2$ and the one that goes on in $q_3$ must accept
Alternating automata

From nondeterministic and universal to alternating automata

Let \( Q' \subseteq Q \)

- Nondeterministic: \( \delta(q_1, \sigma) = \bigvee_{q_i \in Q'} q_i \)
- Universal: \( \delta(q_1, \sigma) = \bigwedge_{q_i \in Q'} q_i \)
- Alternating: Combine the 2 possibilities, allow arbitrary positive boolean formulas
  - “positive”: Don’t use “\( \neg \)"
Tree Automata

Read trees instead of words
- Symbols may have more than one successor, but finitely many
- Automaton forks much like universal word automaton:
  - One copy per child
  - All copies must accept
- But...
  - Each child-automaton runs on a different subtree, not on same input
- Nondeterminism
  - Definition remains
  - Automaton selects possible set of successor-states, then forks and copies run on elements of chosen successor set
Assume finite, binary input tree over $\Sigma = \{a, b, c\}$:

```
      a
     / \
    b   c
```

Automaton $\mathcal{A} = (Q, i_0, \delta, c)$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $i_0 = q_0$, $c$: State in $F = \{q_4\}$ is reached.

Some parts of deterministic tree automaton:

$$
\begin{align*}
\delta: & (q_0, a) \mapsto (q_1, q_2) \\
& (q_1, b) \mapsto (q_4) \\
& (q_2, c) \mapsto (q_4)
\end{align*}
$$

Example for nondeterministic case:

$$
\delta(q_0, a) = \{(q_1, q_2), (q_3, q_2)\}$$
Acceptance

Acceptance conditions for tree automata similar to those of word-automata:

- Final states for finite case
- Büchi, Muller, Rabin, Street or Parity acceptance condition for infinite case
Alternating tree automata

Combination of alternating automata and tree automata not obvious:

- They run on trees
- They allow arbitrary positive boolean expressions for successors...
- ...combined with information about which branch to take
- Branches are enumerated, starting with 0
- Reconsidering the previous example, we can construct an alternating tree automaton out of a “normal” tree automaton:

\[
\delta(q_0, a) = (q_1, q_2) \text{ becomes } \delta(q_0, a) = (0, q_1) \land (1, q_2)
\]

\[
\delta(q_0, a) = \{(q_1, q_2), (q_3, q_2)\} \text{ becomes }
\delta(q_0, a) = (0, q_1) \land (1, q_2) \lor (0, q_3) \land (1, q_2)
\]
Another, partial example:

$$\delta(q_1, \sigma) = (0, q_2) \land (0, q_3) \lor (0, q_3) \land (1, q_3) \land (1, q_4)$$

If you look at the left part...

- It universally branches for the “\land”, i.e. 2 automata are sent into subtrees.
- One descends to the left and starts there in state $q_2$. The other also goes to the left, but into state $q_3$.

As you can see in this example...

- Several copies may proceed in the same subtree
- Subtrees may be ignored

But all running copies of a universal branch must accept!
Theorem (taken from [5]): Given a CTL* formula \( \varphi \) over a set \( AP = I \cup E \cup O \) of atomic propositions and a set \( \tau = 2^{I \cup E} \) of directions, there exists an alternating Rabin tree automaton \( A_{\tau,\varphi} \) over \( 2^{AP} \)-labeled \( \tau \)-trees, with \( 2^O(|\varphi|) \) states and two pairs, such that \( L(A_{\tau,\varphi}) \) is exactly the set of trees satisfying \( \varphi \).

- “Two pairs” refers to the Rabin-acceptance-condition
Overview

- Repetition:
  - Signals $I$ of readable input
  - Signals $E$ of unreadable input
  - Signals $O$ of output

- Since $P$ doesn’t know $E$, it must behave independently of $E$

- If the history of 2 states $p$ and $q$ differs only in values in $E$, then $P$ must behave identical in $p$ and $q$

- However, the signals $E$ are reflected in the computation tree of $P$
**hide**- and **wide**-functions

- **hide** removes the information that is invisible to $P$
  - $\text{hide}_Y(X, Y) = X$
  - We can apply **hide** to a path in a tree by applying it to each node on that path. This yields $\text{hide}_Y : (X \times Y)^* \rightarrow X^*$

- **wide** defines the other direction, but builds consistently labelled trees:
  - $\text{wide}_Y(\langle X^*, V \rangle) = \langle (X \times Y)^*, V' \rangle$ where for every node $w \in (X \times Y)^*$, we have $V'(w) = V(\text{hide}_Y(w))$
Example: *hide-* and *wide-*functions

Consider this 4-ary tree. Assume the first input is $i_0 \in I$ and the second is $e_0 \in E$. Assume arbitrary, potentially inconsistent labels.
Example: *hide*- and *wide*-functions

Hide extracts the binary *I*-part out of the 4-ary tree. Entire subtrees “fall off”
Example: *hide*- and *wide*-functions

- The result looks like this.
- Based on this, *wide* yields a consistently labelled tree.
- That tree still lacks the input signals in the labels, so we need another function.
The *xray*-function adds a labelled tree’s (skeletal) structure to it’s labels:

$$\begin{array}{c}
\text{a} \\
0 \quad 1 \\
\text{b} \quad \text{c}
\end{array} \xrightarrow{\text{xray}} \begin{array}{c}
\langle \varepsilon, \text{a} \rangle \\
0 \quad 1 \\
\langle 0, \text{b} \rangle \quad \langle 1, \text{c} \rangle
\end{array}$$
Overview of automata transformations

- From specification (logic formula $\varphi$), we get Automaton $A$ over $2^{I \cup E \cup O}$ labelled $2^{I \cup E}$ trees.
- A tree accepted by this automaton does not have to be
  - consistent w.r.t. incomplete information.
  - $2^{I \cup E}$ exhaustive.
- So we must construct some automaton $A'$ over $2^O$-labelled $2^{I \cup E}$-tree out of $A$, s.t. $A'$ accepts a tree $\langle T, V \rangle$ iff $A$ accepts $\text{xray}(\langle T, V \rangle)$.
- Then, we still have to deal with incomplete information, so we construct an automaton $A''$ over $2^O$-labelled $2^I$-trees out of $A'$, s.t. $A''$ accepts a tree $\langle T, V \rangle$ iff $A'$ accepts $\text{wide}_{2^E}(\langle T, V \rangle)$. 

Andreas Augustin
Synthesis under incomplete information
Theorem (taken from [1]): Given an alternating tree automaton $A$ over $(\tau \times \Sigma)$-labelled $\tau$-trees, we can construct an alternating tree automaton $A'$ over $\Sigma$-labelled $\tau$-trees such that

1. $A'$ accepts a labelled tree $\langle \tau^*, V \rangle$ iff $A$ accepts $xray(\langle \tau^*, V \rangle)$.
2. $A'$ and $A$ have the same acceptance condition.
3. $|A'| = O(|A|)$
Theorem (taken from [1]): Let $X$, $Y$ and $Z$ be finite sets. Given an alternating tree automaton $A$ over $Z$-labelled $(X \times Y)$-trees, we can construct an alternating tree automaton $A'$ over $Z$-labelled $X$-trees such that

1. $A'$ accepts a labelled tree $\langle X^*, V \rangle$ iff $A$ accepts $\text{wide}_Y(\langle X^*, V \rangle)$.
2. $A'$ and $A$ have the same acceptance condition.
3. $|A'| = O(|A|)$
Solution

- Given $A''$, we can test whether $\mathcal{L}(A'')$ is empty
- $\varphi$ is realizable iff $A''$ is not empty
- The emptiness-check can be extended s.t. it actually produces a finite state program $P$.

Theorem (taken from [1]): The synthesis problem for LTL and $CTL^*$, with either complete or incomplete information, is $2\text{EXPTIME}$ complete.
Final Statements

- We saw that alternation is an appropriate mechanism to cope with incomplete information.
- Something that was not shown here: For the special case of CTL formulas, the algorithm is modifiable, s.t. the obtained algorithm runs in exponential time.
- An extension of the presented result is that $\mu$-calculus synthesis under incomplete information is EXPTIME complete[2], but the extension is not straightforward.
Questions?
References


