Synthesis under incomplete information

Andreas Augustin

June 12, 2008

Overview

Background: Open systems

- 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:
  $P : (2^I)^* \longrightarrow 2^O$

Specification and synthesis

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

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

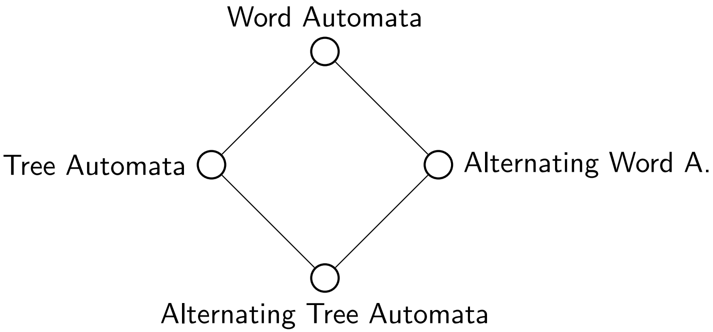
From complete to incomplete information

- 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:
  - $I = \{i\}$; $i = 1 \Leftrightarrow$ User wants to print a paper
  - $E = \{e\}$; $e = 1 \Leftrightarrow$ Paper is buggy
  - $O = \{o\}$; $o = 1 \Leftrightarrow$ Paper scheduled for printing
- We want $A\Box(o \Rightarrow i \wedge \neg e)$
- Since we can't destinguish between $i \wedge \neg e$ and $i \wedge e$, the only safe way to handle this is never to print anything at all

Word- and Tree-Automata and their alternating versions



Word automata

- Well known
  - Alphabet Σ
  - States Q
  - Initial state(s) $i_0 \in Q$ or $I \subseteq Q$
  - Transition-relation or -function δ, details follow
  - Acceptance condition c
- δ may vary depending on the type of atomaton, 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 δ is a function $\delta : Q \times \Sigma \to Q$
- Nondeterministic. Then δ is a relation $\delta : Q \times \Sigma \to 2^Q$
  - Instead of writing $\delta(q_1, \sigma) = \{q_2, q_3\}$ we can write $\delta(q_1, \sigma) = q_2 \vee q_3$ in the sense that the automaton accepts if proceeding in q_2 **or** q_3 accepts
- Universal. Then again, δ is a relation $\delta : Q \times \Sigma \to 2^Q$, but the automaton forks for each additional successor and we demand that all automatons accept
  - Again, we can write $\delta(q_1, \sigma) = q_2 \wedge q_3$, 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 "$\neg$"

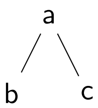
Tree Automata

Read trees instead of words

- Symbols may have more than one successor, but finitely many
- Atomaton forks much like universal word atomaton:
    - 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

Example

- Assume finite, binary input tree over $\Sigma = \{a, b, c\}$:



- 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:
  δ: $(q_0, a) \mapsto (q_1, q_2)$
  $(q_1, b) \mapsto (q_4)$
  $(q_2, c) \mapsto (q_4)$
- 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 finate 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:
  - $\delta(q_0, a) = (q_1, q_2)$ becomes $\delta(q_0, a) = (0, q_1) \wedge (1, q_2)$
  - $\delta(q_0, a) = \{(q_1, q_2), (q_3, q_2)\}$ becomes $\delta(q_0, a) = (0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_2)$

Alternating tree automata

- Another, partial example:
  $\delta(q_1, \sigma) = (0, q_2) \wedge (0, q_3) \vee (0, q_3) \wedge (1, q_3) \wedge (1, q_4)$
- If you look at the left part...
  - It universally branches for the "$\wedge$", 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!

$\varphi \to \mathcal{A}$

Theorem (taken from [5]): *Given a CTL* formula φ 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 $\mathcal{A}_{\tau,\varphi}$ over 2^{AP}-labeled τ-trees, with $2^{O(|\varphi|)}$ states and two pairs, such that $\mathcal{L}(\mathcal{A}_{\tau,\varphi})$ is exactly the set of trees satisfying φ.*

- "Two pairs" refers to the Rabin-acceptance-condition

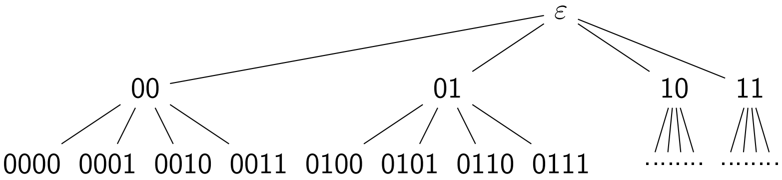
Overview

- 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

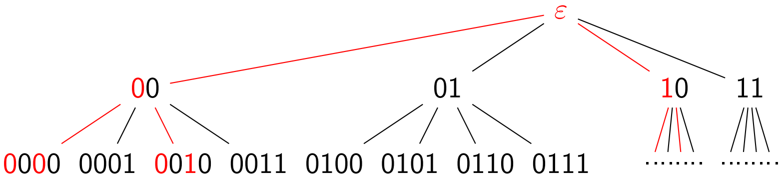
- *hide* removes the information that is invisible to P
  - $hide_Y(X, Y) = X$
  - We can apply *hide* to a path in a tree by applying it to each node on that path. This yields $hide_Y : (X \times Y)^* \to X^*$
- *wide* defines the other direction, but builds consistently labelled trees:
  - $wide_Y(\langle X^*, V\rangle) = \langle (X \times Y)^*, V'\rangle$ where for every node $w \in (X \times Y)^*$, we have $V'(w) = V(hide_Y(w))$

Example: *hide*- and *wide*-functions



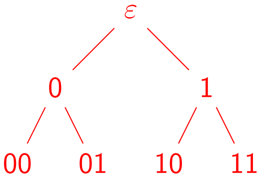
Consider this 4-ary tree. Assume the first input is $i_0 \in I$ and the second is $e_0 \in E$. Assume arbitrary, potentially inconsistent labels

Example: *hide*- and *wide*-functions



Hide extracts the binary *I*-part out of the 4-ary tree. Entire subtrees "fall off"

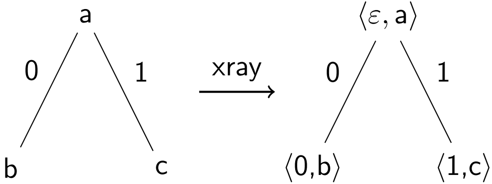
Example: *hide*- and *wide*-functions



- 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

The *xray*-function adds a labelled tree's (skeletal) structure to it's labels:



Overview of automata transformations

- From specification (logic formula φ), we get Automaton $\mathcal{A}$ 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 $\mathcal{A}'$ over 2^O-labelled $2^{I \cup E}$-tree out of $\mathcal{A}$, s.t. $\mathcal{A}'$ accepts a tree $\langle T, V \rangle$ iff $\mathcal{A}$ accepts $xray(\langle T, V \rangle)$
- Then, we still have to deal with incomplete information, so we construct an automaton $\mathcal{A}''$ over 2^O-labelled 2^I-trees out of $\mathcal{A}'$, s.t. $\mathcal{A}''$ accepts a tree $\langle T, V \rangle$ iff $\mathcal{A}'$ accepts $wide_{2^E}(\langle T, V \rangle)$

$\mathcal{A} \to \mathcal{A}'$

Theorem (taken from [1]): *Given an alternating tree automaton $\mathcal{A}$ over $(\tau \times \Sigma)$-labelled τ-trees, we can construct an alternating tree automaton $\mathcal{A}'$ over Σ-labelled τ-trees such that*

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

$\mathcal{A}' \rightarrow \mathcal{A}''$

Theorem (taken from [1]): *Let X, Y and Z be finite sets. Given an alternating tree automaton $\mathcal{A}$ over Z-labelled $(X \times Y)$-trees, we can construct an alternating tree automaton $\mathcal{A}'$ over Z-labelled X-trees such that*

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

Solution

- Given $\mathcal{A}''$, we can test whether $\mathcal{L}(\mathcal{A}'')$ is empty
- φ is realizable iff $\mathcal{A}''$ 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 2EXPTIME complete.*

Final Statements

- We saw that alternation is an apropriate machanism 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 μ-calculus synthesis under incomplete information is EXPTIME complete[2], but the extension is not straightforward.

Questions?

References

[1] Main paper: Orna Kupferman, Moshe Y. Vardi. Synthesis with incomplete information.

[2] Broader overview: Orna Kupferman, Moshe Y. Vardi. μ-calculus synthesis.

[3] LTL, CTL, Alternating tree automata: Moshe Y. Vardi. Alternating automata and program verification.

[4] S1S: Madhavan Mukund. Finite-state automata on infinite inputs.

[5] From Logics to alternating automata: O. Bernholtz, M. Y. Vardi and P. Wolper. An automata-theoretic approach to branching-time model checking