Synthesis under incomplete information

Andreas Augustin

June 12, 2008

Contents

1 Outline

1.1 Open systems

We know automata that read input and make transitions for both, the finite and the infinite case. In basic lectures, there's an introduction to automata, that read input, produce output and make transitions, too. The most prominent examples are Moore and Mealy automata. Those come close to the behaviour of a reactive system, which is basically a program P that maps inputs I and the history of previously seen inputs to outputs O: $P : (2^I)^* \to 2^O$. P is state-based and the history is available only in the states.

1.2 Specification and synthesis

The specification of a program can be done in form of a formula φ, e.g. in LTL, CTL, CTL* or μ-calculus. Given such a formula φ, realizability corresponds to the question, whether there

exists a program P that satisfies φ and synthesis is the problem to transform the specification φ into a program P that is guaranteed to satisfy φ.

1.2.1 Synthesis for LTL

For LTL, the specification yields allowed combinations of sequences of inputs and outputs and the whole problem can be reduced to a non-emptiness test of a tree-automaton, that can be constructed out of the specification. Synthesis is proven to be 2EXPTIME complete in this case.

1.2.2 Synthesis for branching-time logics

For branching-time logics like CTL and CTL*, P associates with each input sequence an infinite computation over $2^{I \cup O}$.[1] Although P deterministic, P induces a computation tree due to external nondeterminism caused by different possible inputs in I. Branching temporal logics give us the required expressive power because of path quantifiers: In LTL we can't express possibility requirements. And again, realizability correlates to a non-emptiness-test for a tree-atomaton that can be constructed out of the specification.

1.3 Incomplete information

For incomplete information, let's now assume that the environment knows more than the program P: This can be modelled using signals E that are known to the environment, but unknown to P in addition to the signals I of readable input and signals O of output.

This directly leads to the question how big the impact on realizability and complexity is, compared with synthesis under complete information.

2 Automata types

Let's first have a look at the required automata: We'll go from the well-known word automata over both, alternating word and non-alternating tree-automata to alternating tree automata.

2.1 Word automata

Word automata are considered to be well-known for the purpose of this summary. Notationally, we assume an automaton to consist of an alphabet Σ, states Q, an initial state $i_0 \in Q$ or several initial states $I \subseteq Q$, a transition-relation or -function δ and an acceptance condition c. δ may vary depending on the type of atomaton, determinism and so forth. c may be something like Muller-Acceptance or Rabin-Acceptance.
A word automaton can be...

- Deterministic. Then δ is a function $\delta : Q \times \Sigma \to Q$.
- Nondeterministic. Then δ is a relation $\delta : Q \times \Sigma \to 2^Q$.
- 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.

[1] Since I and O are disjoint, $2^{I \cup O} = 2^I \times 2^O$ holds. Both notations are commonly used the literature.

Instead of writing $\delta(q_1, \sigma) = \{q_2, q_3\}$ we can write $\delta(q_1, \sigma) = q_2 \vee q_3$ for the nondeterministic case and $\delta(q_1, \sigma) = q_2 \wedge q_3$ for the universal case. This can be understood like follows: The automaton spawns copies, the copies proceed into the succeeding states (q_2 and q_3 in the example), and for the nondeterministic case, the automaton accepts if copy 1 **or** copy 2 accepts and in the universal case, it acceps only if both copies, the copy proceeding in q_2 **and** the copy proceeding in q_3 accept.

Since several copies may proceed from a single node, the computation of the automaton spans up a tree, a computation tree.

2.2 From nondeterministic and universal to alternating automata

For nondeterministic and universal branching, we restricted ourselves to using either "or" or "and". We could use more than one "$\vee$" in a nondeterministic formula, e.g. $\delta(q_1, \sigma) = q_2 \vee q_3 \vee q_4$, but in a nondeterminstic automaton, we're not allowed to use "$\wedge$" and in a universal automaton, we're not allowed to use "$\vee$".

For the alternating automaton, we now combine the 2 possibilities, but we still forbid "$\neg$". Therefore, we allow arbitrary positive boolean formulas.

2.3 Tree Automata

Tree Automata read trees instead of words. In an input tree, a symbol may have more than one successor, but still only finitely many. In this sense, a word is a tree in that each symbol has at most one successor. If we consider infinite words, we have to furtheron restrict this to "exactly one successor". For a general tree with possibly more than one successor-symbol, the automaton forks much like a universal word atomaton: It spawns one copy per child and all copies must accept. What's new is that each child-automaton runs on a different subtree, not on the same input. We demand that, given a symbol σ with n successors, Let $q \in Q$, $\delta(q, \sigma) \in Q^n$.

We can also construct nondeterministic tree automata, there we allow more than one possible successor-set: $\delta(q, \sigma) \subseteq Q^n$. The automaton selects a possible set of successor-states, then spawns copies of itself and the copies run on the elements of the chosen successor set.

2.4 Acceptance

Acceptance conditions for tree automata are similar to those of word-automata, we have final states for the finate case and acceptance condisions like Büchi, Muller, Rabin, Street or Parity acceptance for the infinite case. For the infinite case, we demand that the acceptance condition is met for all paths from each node in the computation tree backwards to the root of the tree, e.g. each such path must contain infinitely many final states for a Büchi acceptance condition to hold.

2.5 Alternating tree automata

Like non-alternating tree automata, alternating tree automata run on trees and like alternating word automata, they allow arbitrary positive boolean expressions for successors, but this time these expressions are combined with information about which branch to take. Therefore, branches are enumerated, starting with 0, or more generally, we have **directions**. For each

node, there's a unique direction for each of its children. When using numbers, we speak of direction 0, direction 1 a.s.f.

For example, $\delta(q_0, a) = (q_1, q_2)$ becomes $\delta(q_0, a) = (0, q_1) \wedge (1, q_2)$,
and $\delta(q_0, a) = \{(q_1, q_2), (q_3, q_2)\}$ becomes $\delta(q_0, a) = (0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_2)$.

Several copies may proceed in the same direction, while other directions may be ignored entirely. But still, all running copies of a universal branch must accept!

3 Incomplete information

To get back to incomplete information, let's have a look at the following theorem:
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[2], such that $\mathcal{L}(\mathcal{A}_{\tau,\varphi})$ is exactly the set of trees satisfying φ.*

Unfortunately, there are still 2 problems to solve:

1. Although the input trees are τ-trees, they needn't be $I \cup E$-exhaustive, i.e. when going into direction ie, we may find a set of atomic propositions that is completely independent of ie. What we want is that, given i and e, if we go into direction ie, then we want to find the propositions for i and e there, such that we can derive an output o for i and e by going into direction ie.

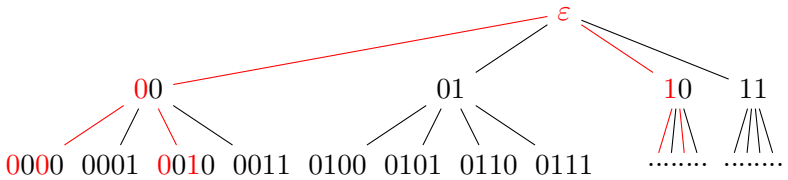
2. Since P doesn't know E, it must behave independently of E. If the history of 2 states p and q differs only in values in E, then P must behave identical in p and q. However, the signals E are reflected in the computation tree of P. So we have to make sure that P still behaves independently of them.

3.1 hide, wide, xray and cover functions

To solve these problems, we introduce some functions. The *hide*- and *wide*-functions help us to get rid of inconsistencies w.r.t signals in E and *xray* and *cover* help us to deal with the problem of trees possibly not being $I \cup E$-exhaustive.

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: Consider this 4-ary tree. Assume the first input is $i \in I$ and the second is $e \in E$. Assume arbitrary, potentially inconsistent labels. 2 binary signals $i \in I$ and $e \in E$ span up a 4-ary tree. Visible to P is only the binary, red part of the tree:

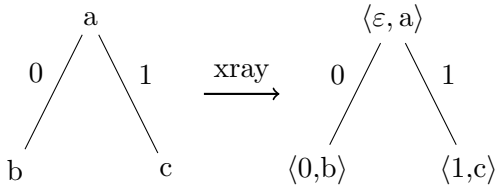


Hide extracts the binary I-part out of the 4-ary tree. Entire subtrees "fall off". Based on this

[2] "Two pairs" refers to the Rabin-acceptance-condition.

result, *wide* yields a consistently labelled tree, e.g. the nodes in directions 00 and 01 must carry the same label, since they only differ in e.

Anyway, we haven't dealt yet with $I \cup E$-exhaustiveness. Therefore, we introduce the *xray*-function: The *xray*-function adds a labelled tree's (skeletal) structure to it's labels:



Similarly to *hide* and *wide*, *cover* reverses *xray*: $cover(xray(\tau)) = \tau$.

3.2 Putting it all together

1. From a specification (logic formula φ), we get an 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, nor does it have to be $I \cup E$ exhaustive
2. So we must construct some automaton $\mathcal{A}'$ over 2^O-labelled $2^{I \cup E}$-trees out of $\mathcal{A}$, s.t. $\mathcal{A}'$ accepts a tree $\langle T, V \rangle$ iff $\mathcal{A}$ accepts $xray(\langle T, V \rangle)$.
3. 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 $wide_{2^E}(\langle T, V \rangle)$

All these goals are achievable, as stated in the following theorems:

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. $|\mathcal{A}'| = O(|\mathcal{A}|)$

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. $|\mathcal{A}'| = O(|\mathcal{A}|)$

3.3 Solution

Given $\mathcal{A}''$, we can test whether $\mathcal{L}(\mathcal{A}'')$ is empty. φ is realizable iff $\mathcal{A}''$ is not empty and the emptiness-check can be extended s.t. it actually produces a finite state program P. Given the 2 previous theorems, we can already guess that the given transformations didn't make the process more complex and the following theorem confirms this intuition:

Theorem (taken from [1]): *The synthesis problem for LTL and CTL*, with either complete or incomplete information, is 2EXPTIME complete.*

3.4 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 as straightforward as the extension for CTL.

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