Synthesis with Incomplete Information

Orna Kupferman, Moshe Vardi

2nd International Conference on Temporal Logic, 1997

Michael Greenberg
Technion, Spring 2006

Introduction

- What is verification?
  - Check whether program P specification ψ
    - Potentially use counterexamples to refine and recheck
- When do we get ψ and P?

Program synthesis

- *synth* : specification $\psi \rightarrow$ program P
  - such that P ψ
- What is ψ?
  - Straight-line vs. reactive
- Temporal logic

Temporal logic for program synthesis

- Linear or branching time?
  - CTL describes trees, viz. unfolded programs
  - LTL describes words, viz. computations
- CTL^*
  - General method using standard constructions
  - *synth*(ψ) : $CTL^* \rightarrow$ program P
    - (s.t. P ψ)

What are our specifications?

- Quantification
  - x is the input
  - y is the output

  $\forall x \forall y.\psi$? $\forall x \exists y.\psi$? $\exists x \forall y.\psi$? $\exists x \exists y.\psi$?
- Complete, open module: $\forall x \exists y.\psi$
  - In fact: $\forall x \exists y.A\psi$
- Is every $\psi \in$ CTL* *realizable*?
  - $\psi = x \quad \neg x$

A money-making proposal

Step 1: $\psi = G(Fx \Leftrightarrow y)$

Step 2: ???

Step 3: Profit!

- What is wrong with $\forall x \exists y. AG(Fx \Leftrightarrow y)$?
  - Satisfiable: let y be Fx.
  - Valid: is satisfiable and has no free variables.

The program synthesis hydra

- Two problems
  - Identification of *realizability*
  - Realization (read “program synthesis”)

Realizability

- A formula $\psi \in CTL^*$ is realizable *iff* there exists a program P such that P ψ
  - Any (deterministic) program P can be seen as a *strategy function* $P : (2^I)^* \rightarrow 2^O$

Incomplete information

- With “incomplete information”, there are two disjoint sets of input signals, I and E
  - I is known
  - E is unknown
    - Iterative realizability checks
    - I   E = S
- Strategy functions see only I

Informal discussion

- **Trees**
- Tree automata
- Emptiness checks
- Program synthesis

Trees

- For a finite set Y, an Y-tree is a set $T \subseteq Y$
  - s.t. $x \cdot v \in T \quad v \in Y \Rightarrow x \in T$
- $\mathit{dir}(x \cdot v) \equiv v$; $\mathit{dir}(\varepsilon) = v^0$
- T is a *full infinite tree* iff $T = Y^*$
- For finite sets Y and Σ, a Σ-labeled Y-tree is a pair <T, V>
  - T is an Y-tree
  - $V : T \rightarrow \Sigma$

Tree operators: *x-ray*

- *x-ray*(<T, V>) is an (Υ x Σ)-labeled Υ-tree <T', V'>
  - s.t. V'(x) = <*dir*(x), V(x)>

Tree operators: *hide* and *wide*

- $hide_Y(T \subseteq (X \times Y)^*) = T' \subseteq X^*$ replacing each letter <x, y> with the letter x
- $wide_Y(\langle T \subseteq X^*, V\rangle) = \langle T', V'\rangle$
  - s.t. $T' \subseteq (X \times Y)^*$ is a full infinite tree and $V'(w) = V(hide_Y(w))$

Tree operators: *fat*

- fat_y is a generalization of $wide_Y$
- $fat_y(<T \subseteq X^*, V>) = \{ <T' \subseteq X^* \times Y^*, V'> \mid$

  $V'(\varepsilon) \quad X = V(\varepsilon)$

  $\forall w \in (X^* \times Y^*)^+ .\ V'(w) = V(hide_Y(w)) \quad (dir(w) \quad E) \}$

- $wide'_y(<T,V>) \in fat_y(<T,V>)$
  - $V'(\varepsilon) = V(\varepsilon)$
  - If V is X-exhaustive:
    - $wide'_y(<T,V>) = x\text{-}ray(wide_y(<T,V>))$

Computation trees

- Given a full infinite 2^{I}-tree, a program P induces a 2^{O}-labeled tree $<(2^{I})^{*}, P>$
- Add in E: $<(2^{I\quad E})^{*}, P'> = \mathit{wide}_{(2^{E})}(<(2^{I})^{*}, P>)$
- A computation tree is the $2^{I\quad E\quad O}$-labeled tree $<(2^{I\quad E})^{*}, P''> = \textit{x-ray}(<(2^{I\quad E})^{*}, P'>)$

Informal discussion

- **Trees**
- **Tree automata**
- Emptiness checks
- Program synthesis

Tree automata

- Seen during Yoram and Shahar's presentation of CTL automata
- Alternating form equivalent to $\mathscr{L}\mu$ [EJ91]
  - CTL^* is contained, of course
- $\delta : Q \times \Sigma \rightarrow \mathit{B}^+(Y \times Q)$
  - A state has transitions for a given input label
  - Boolean formula over the tree direction and next states

CTL* automata

- Automaton $A_{Y,\psi}$ from $\psi \in CTL^*$ and a set Y
  - $O(2^{|\psi|})$ states
  - 2-pair acceptance condition
- Accepts computation trees
  - 2^{AP}-labeled Y-trees $<T, V>$
    - $AP = I \quad E \quad O$
    - $Y = 2^{I \quad E}$
  - $\mathscr{L}(A_{Y,\psi}) = \{<T, V> \mid <T, V> \quad \psi\}$

Michael Greenberg
Technion, Spring 2006

Informal discussion

- **Trees**
- **Tree automata**
- **Emptiness checks**
- Program synthesis

Our problem, formally

- Given a CTL* formula ψ over AP = I   E   O (disjoint), is there a program P such that its computation tree satisfies ψ?
- Given a tree automaton A accepting only trees that satisfy ψ, is $\mathscr{L}$(A) empty?
  - Realizability only?

Emptiness (informally)

- $\mathscr{L}(A) \neq \emptyset \Leftrightarrow \exists$<T,V> s.t. A accepts <T,V>
  - A accepts <T,V> $\Leftrightarrow \exists$ a run of A on <T,V> that accepts
  - This run makes "determinations"
- Given regular determinations, we have a deterministic tree automaton on Y-trees labeled by Σ
  - $Y = 2^I$
  - $\Sigma = 2^O$

Informal discussion

- **Trees**
- **Tree automata**
- **Emptiness checks**
- **Program synthesis**
- **~~Break!~~**
- **Program synthesis**

The synthesis theorem

Given an alternating tree automaton A over Σ-labeled Y-trees, the following are equivalent:

- A is nonempty.
- There is a finite-state strategy $f : Y^* \to \Sigma$
  - s.t. $\langle Y^*, f \rangle \in \mathscr{L}(A)$

Nonemptiness algorithms can be extended to generate this strategy.

Programs?

- Emptiness checks can generate “programs”
  - Input alphabet: $Y = 2^{I \quad E}$
  - Output alphabet: $\Sigma = 2^{AP} = 2^{I \quad E \quad O}$
- Real program
  - $Y = 2^{I}$
  - $\Sigma = 2^{O}$
- Never mind that $A_{Y,\psi}$ is alternating...

Automaton labels

- Given $A_{Y,\psi}$, generate A':
  - $Y = 2^{I \quad E}$
  - $\Sigma = 2^{O}$
  - $\mathscr{L}(A') = \mathscr{L}(A_{Y,\psi})$ projected onto 2^{O}
- O(|A|) transformation *cover*(A)
  - A' accepts $<Y^*,V> \Leftrightarrow$
    A accepts *x-ray*($<Y^*,V>$)

cover($A_{Y,\psi}$)

- $A_{Y,\psi}$ has alphabet $2^{I \quad E \quad O} = Y \times \Sigma$
  - A' wants alphabet $2^O = \Sigma$
- $A_{Y,\psi}$ has state-set Q
  - A' can have Q x Y
- Record ($\upsilon \in Y$) directions in each state of A'
  - Start state $\langle q_0, v^0 \rangle$
  - $\delta'(\langle q,\upsilon \rangle, \sigma) = \delta(q, \langle \upsilon,\sigma \rangle)$
    - Changing (υ', q') in δ to (υ', $\langle q',\upsilon' \rangle$) in δ'
  - q is accepted $\Rightarrow$ (q, υ) is accepted; and v.v.

Pruning the tree (automaton)

- $Y = 2^{I \quad E}$
  - But the program shouldn't see unknown events
- Automaton A over Σ-labeled (X x Y)-trees, generate $narrow_Y$(A) = A' over Σ-labeled X-trees:
  - A' accepts $<X^*,V>$ ⇔ A accepts $wide_Y(<X^*,V>)$
  - Simple:
    - Q = (X x Y); Q' = X
    - δ'(q, z) = δ(q, z) replacing (<x,y>, q') with (x, q')

Quick proof of $narrow_Y$

- $wide_Y(<X^*,V>) \in \mathscr{L}(A) \Rightarrow <X^*,V> \in \mathscr{L}(A')$
  - States the same
  - Pick an accepting run, drop the Y component
- $<X^*,V> \in \mathscr{L}(A') \Rightarrow wide_Y(<X^*,V>) \in \mathscr{L}(A)$
  - Define A'' with states Q x Y marking Y direction
    - $\mathscr{L}(A'') = \mathscr{L}(A)$
  - A run through states in A'' can be adjusted to a run in A

Program synthesis

- Create $A_{Y,\psi}$ over $2^{I \quad E \quad O}$-labeled $2^{I \quad E}$-trees
- Compute $A'' = \mathit{narrow}_{(2^E)}(\mathit{cover}(A_{Y,\psi}))$ over 2^O-labeled 2^E-trees
  - $<(2^I)^*,P> \in \mathscr{L}(A'') \Rightarrow P \quad \psi \Rightarrow \psi$ realizable
  - ψ realizable $\Rightarrow$
    $\exists$ a computation tree $\in \mathscr{L}(A_{Y,\psi}) \Rightarrow$
    $\mathit{x\text{-}ray}(\mathit{wide}_{(2^E)}(<(2^I)^*,P>)) \in \mathscr{L}(A_{Y,\psi}) \Rightarrow$
    $\mathit{wide}_{(2^E)}(<(2^I)^*,P>) \in \mathscr{L}(\mathit{cover}(A_{Y,\psi})) \Rightarrow$
    $<(2^I)^*,P> \in \mathscr{L}(\mathit{narrow}_{(2^E)}(\mathit{cover}(A_{Y,\psi})))$
- Constructive emptiness check

Formally...

Formal Discussion

- **A fixpoint-based emptiness check**
- CTL program synthesis

Emptiness

- $O((mn)^{3n})$ for nondeterministic tree automata [EJ88]
  - $m = |A|$
  - n = number of pairs in Rabin acceptance condition Γ
- Works by model checking (sort of)
  - $A\Phi_\Gamma = A(\ldots \quad (GF(g_\gamma) \quad FG(\neg b_\gamma)) \quad \ldots)$
    - $i \in \{ 0 \ldots m - 1 \}$

“Model checking” Overview

- Nondeterministic tree automata aren't convenient
- Convert to an AND/OR diagram T, pseudo-model-check
  - |T| is O(|A|)
- $\mu Y. \quad {}_{\gamma \in \Gamma} AFAG((\neg b_\gamma \quad Y) \quad A(Fg_\gamma \quad \Phi_{\Gamma/\{\gamma\}}))$
  - Φ_Γ also adjusted
  - $T,s \quad A\Phi_\Gamma \Leftrightarrow T,s \quad \exists i.Y^i, i \leq |T|$

“Model checking” $A(Fg_\gamma \quad \Phi_{\Gamma/\{\gamma\}})$

- LHS: T,s AFq ⇔ T,s μx.(q EXAXx)
- $val(T, AFg_\gamma) \quad val(T, A\Phi_{\Gamma/\{\gamma\}})$ doesn't work
  - Make it disjoint manually?
- RHS: $val(T/val(T, AFg_\gamma), A\Phi_{\Gamma/\{\gamma\}})$
  - Recursive in other pairs

“Model checking” $AG(g_\gamma(Y))$

- $g_\gamma(Y) = (\neg B_\gamma \quad Y) \quad A(Fg_\gamma \quad \Phi_{\Gamma/\{\gamma\}})$
- $AGr = \nu x.(r \quad AXx)$
- Determinations from the RHS?
  - $z^k = \nu z.val(z, g_\gamma(Y)) \quad EXAX(z)$
    - z is initially $val(T, g_\gamma(Y))$
  - $T,s \quad AGg_\gamma(Y^i) \Rightarrow$

    $s \in z^k \Rightarrow$

    $T,s \quad \exists\alpha.AGg_\gamma(Y^\alpha)$

Finishing “Model checking”

- $\mu x.(val(T, AG(g_\gamma(Y)))\quad EXAXx)$
  - $T,s\quad AFAGg_\gamma(Y^i) \Rightarrow$
    $s \in val(T, AFAGg_\gamma(Y^i)) \Rightarrow$
    $T,s\quad \exists\alpha.AFAGg_\gamma(Y^\alpha)$
- Do all of that for each γ
  - $O(|\Gamma||T|^2)$ sub-checks for $|\Gamma|-1$ pairs
  - Total work: $O((|\Gamma||T|)^{3|\Gamma|})$

The delightful side-effect

- A good thing: our final fixpoint is a deterministic subgraph satisfying ψ!
  - A “Hintikka structure”, necessitated by the Small Model Theorem
- Extracted Hintikka structures can be massaged into “transducers”
  - From I to O: a program!

Formal Discussion

- **A fixpoint-based emptiness check**
- **CTL program synthesis**

CTL as $\mathscr{L}_{\text{spec}}$

- CTL is very widely used and understood
- Clearer translation
- Simpler than CTL^* $\Rightarrow$ improved complexity
  - Less expressive...

The automaton

- A_ψ over $2^{I\quad O}$-labeled 2^I-trees
  - $O(|\psi|)$ states
    - q_0, $\{\, cl(\psi) \times \{\, \exists, \forall \,\} \,\}$
  - Büchi acceptance condition
    - A_ψ accepts $\langle T,V\rangle \Leftrightarrow \mathit{wide}'_{2^E}(\langle T,V\rangle) \quad \psi$
  - Transition function δ as in Yoram and Shahar's presentation
    - Adjusted for modes

The pre-transition relation δ'

- $\delta'(\quad,\sigma) : \mathit{cl}(\psi) \text{ x } 2^{I \quad E \quad O} \rightarrow B^{+}(2^{I} \text{ x } Q)$
  - $\delta'(p \in I \quad E \in O, \sigma) = p \quad \sigma$
  - Logical connectives carry through
  - MX becomes (, M)
    - over inputs for E, over inputs for A
  - $M[\ _1 \text{ U } \ _2]$ becomes the usual or/recursive and
    - Same input connectives as X
  - MG is the same as M[true U]

The transition relation δ

- For $\in$ *cl*(ψ) and $\upsilon \in 2^{I \quad O}$
  - $\delta(<, \quad M>, \upsilon) = \delta'(\quad , \upsilon \quad \tau)$
    - M = ∃: over all τ in 2^E
    - M = ∀: over all τ in 2^E
- $\delta(q_0, \upsilon) = \delta'(\psi, \upsilon)$
- Some reductions:
  - p ∈ E can be reduced to true/false for ∃/∀
  - EX and AX are δ' regardless of state mode
    - $\delta(<EX \quad , M>, \upsilon) = \delta'(EX \quad , \upsilon)$

2^I-exhaustiveness

- Recall: A_ψ accepts $<T,V> \Leftrightarrow \textit{wide}'_2E(<T,V>) \quad \psi$
- V need not be 2^I-exhaustive
  - i.e. $V(w) \quad 2^I \quad \textit{dir}(w) \quad 2^I$
  - A_ψ accepts incomplete computation trees
- Good news: X-exhaustiveness is regular!
  - Build an automaton A_{exh}

Emptiness for Nondeterministic Büchi Tree Automata

- Büchi condition allows for PTIME method [VW84]
- Finds Hintikka structures

CTL program synthesis

- Generate A_ψ (alternating)
- Cross with A_{exh} (nondeterministic)
- Perform constructive emptiness test
- EXPTIME
  - Exponential blowup on conversion from alternating to nondeterministic
  - Polynomial emptiness check

Discussion

- Useful?
- Tractable?
- Implementation?