Synthesis with Incomplete Information

Orna Kupferman, Moshe Vardi
2nd International Conference on Temporal Logic, 1997
Introduction

- What is verification?
  - Check whether program P specification \( \psi \)
    - Potentially use counterexamples to refine and recheck
- When do we get \( \psi \) and P?
Program synthesis

• $\textit{synth} : \text{specification } \psi \rightarrow \text{program } P$
  – such that $P \models \psi$

• What is $\psi$?
  – 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$(\psi)$ : CTL$^*$ → program P
    • (s.t. P $\models$ $\psi$)
What are our specifications?

- Quantification
  - x is the input
  - y is the output
  \[ \forall x \forall y. \psi \lor \forall x \exists y. \psi \lor \exists x \forall y. \psi \lor \exists x \exists y. \psi \]

- Complete, open module: \( \forall x \exists y. \psi \)
  - In fact: \( \forall x \exists y. A \psi \)

- Is every \( \psi \in \text{CTL}^* \) realizable?
  - \( \psi = x \lor \neg x \)
A money-making proposal

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

Step 2: ???

Step 3: Profit!

- What is wrong with $\forall x \exists y. AG(Fx \iff y)$?
  - 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 $\psi \in \text{CTL}^*$ is realizable \emph{iff} there exists a program $P$ such that $P \models \psi$
  - Any (deterministic) program $P$ can be seen as a \textit{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

Michael Greenberg
Technion, Spring 2006
Informal discussion

• Trees
• Tree automata
• Emptiness checks
• Program synthesis
Trees

- For a finite set $Y$, an $Y$-tree is a set $T \subseteq Y$ such that $x \cdot v \in T \quad v \in Y \Rightarrow x \in T$

- $\text{dir}(x \cdot v) \equiv v$ ; $\text{dir}(\varepsilon) = v^0$

- $T$ is a *full infinite tree* iff $T = Y^*$

- For finite sets $Y$ and $\Sigma$, a $\Sigma$-labeled $Y$-tree is a pair $<T, V>$
  - $T$ is an $Y$-tree
  - $V : T \rightarrow \Sigma$
Tree operators: x-ray

- \textit{x-ray}(<T, V>) is an \((Y \times \Sigma)\)-labeled \(Y\)-tree \(<T', V'>\)
  - s.t. \(V'(x) = <\text{dir}(x), V(x)>\)
Tree operators: *hide* and *wide*

1. \(\text{hide}_Y(T \subseteq (X \times Y)^*) = T' \subseteq X^*\) replacing each letter \(<x, y>\) with the letter \(x\)

2. \(\text{wide}_Y(<T \subseteq X^*, V>) = <T', V'>\)
   - s.t. \(T' \subseteq (X \times Y)^*\) is a full infinite tree and \(V'(w) = V(\text{hide}_Y(w))\)
Tree operators: \textit{fat}

- \textit{fat}_Y is a generalization of \textit{wide}_Y

- \textit{fat}_Y(\langle T \subseteq X^*, V \rangle) = \{ \langle T' \subseteq X^* \times Y^*, V' \rangle \mid
  V'(\varepsilon) = V(\varepsilon)
  \forall w \in (X^* \times Y^*)^+. V'(w) = V(\text{hide}_Y(w)) (\text{dir}(w) E) \}

- \textit{wide}'_Y(\langle T, V \rangle) \in \textit{fat}_Y(\langle T, V \rangle)
  - V'(\varepsilon) = V(\varepsilon)
  - If V is X-exhaustive:
    - \textit{wide}'_Y(\langle T, V \rangle) = \textit{x-ray}(\textit{wide}_Y(\langle T, V \rangle))
Computation trees

- Given a full infinite $2^I$-tree, a program $P$ induces a $2^O$-labeled tree $<(2^I)^*, P>$
- Add in $E$: $<(2^I^E)^*, P'> = wide_{(2^E)}((2^I)^*, P>)$
- A computation tree is the $2^I^E^O$-labeled tree $<(2^I^E)^*, P''> = x-ray((2^I^E)^*, P'>)$
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 $L\mu$ [EJ91]
  - CTL$^*$ is contained, of course
- $\delta : Q \times \Sigma \rightarrow B^+(Y \times Q)$
  - A state has transitions for a given input label
  - Boolean formula over the tree direction and next states
CTL* automata

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

- Trees
- Tree automata
- Emptiness checks
- Program synthesis
Our problem, formally

- Given a CTL* formula $\psi$ over $\text{AP} = \text{I} \quad \text{E} \quad \text{O}$ (disjoint), is there a program $P$ such that its computation tree satisfies $\psi$?

- Given a tree automaton $A$ accepting only trees that satisfy $\psi$, is $\mathcal{L}(A)$ empty?
  - Realizability only?
Emptiness (informally)

- $\mathcal{L}(A) \neq \emptyset \iff \exists <T,V> \text{ s.t. } A \text{ accepts } <T,V>$
  - $A \text{ accepts } <T,V> \iff \exists \text{ a run of } A \text{ on } <T,V> \text{ that accepts}$
  - This run makes “determinations”

- Given regular determinations, we have a deterministic tree automaton on $Y$-trees labeled by $\Sigma$
  - $Y = 2^I$
  - $\Sigma = 2^O$
Informal discussion

- Trees
- Tree automata
- Emptiness checks
- Program synthesis

Break!
- Program synthesis
The synthesis theorem

Given an alternating tree automaton $A$ over $\Sigma$-labeled $\Upsilon$-trees, the following are equivalent:

- $A$ is nonempty.
- There is a finite-state strategy $f : \Upsilon^* \to \Sigma$
  - s.t. $<\Upsilon^*, f> \in \mathcal{L}(A)$

Nonemptiness algorithms can be extended to generate this strategy.
Programs?

• Emptiness checks can generate “programs”
  – Input alphabet: $Y = 2^1$  
  – Output alphabet: $\Sigma = 2^{AP} = 2^1$  
• Real program
  – $Y = 2^1$
  – $\Sigma = 2^O$
• Never mind that $A_{Y,\psi}$ is alternating...
Automaton labels

• Given $A_{\gamma,\psi}$, generate $A'$:
  - $\gamma = 2^I$ \(\forall\)
  - $\Sigma = 2^O$
  - $L(A') = L(A_{\gamma,\psi})$ projected onto $2^O$

• $O(|A|)$ transformation $\text{cover}(A)$
  - $A'$ accepts $<\gamma^*,V> \iff$
    $A$ accepts $x\text{-ray}(<\gamma^*,V>)$
cover(\(A_{Y,\psi}\))

- \(A_{Y,\psi}\) has alphabet \(2^I \ E \ O = Y \times \Sigma\)
  - \(A'\) wants alphabet \(2^O = \Sigma\)
- \(A_{Y,\psi}\) has state-set \(Q\)
  - \(A'\) can have \(Q \times Y\)
- Record (\(u \in Y\)) directions in each state of \(A'\)
  - Start state \(<q_0, v^0>\)
  - \(\delta'(\langle q, u \rangle, \sigma) = \delta(q, \langle u, \sigma \rangle)\)
    - Changing (\(u', q'\)) in \(\delta\) to (\(u', \langle q', u' \rangle\)) in \(\delta'\)
    - \(q\) is accepted \(\Rightarrow\) (\(q, u\)) is accepted; and v.v.
Pruning the tree (automaton)

- $Y = 2^E$
  - But the program shouldn't see unknown events
- Automaton $A$ over $\Sigma$-labeled $(X \times Y)$-trees, generate $\text{narrow}_\gamma(A) = A'$ over $\Sigma$-labeled $X$-trees:
  - $A'$ accepts $<X^*, V> \iff A$ accepts $\text{wide}_\gamma(<X^*, V>)$
- Simple:
  - $Q = (X \times Y)$; $Q' = X$
  - $\delta'(q, z) = \delta(q, z)$ replacing $(<x, y>, q')$ with $(x, q')$
Quick proof of $\text{narrow}_Y$

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

- Create $A_{\Psi,\Psi}$ over $2^I_E$ $O$-labeled $2^I_E$-trees
- Compute $A'' = \text{narrow}_{(2^E)}(\text{cover}(A_{\Psi,\Psi}))$ over $2^O$-labeled $2^E$-trees

- $<(2^I)^*,P> \in \mathcal{L}(A'') \Rightarrow \Psi \Rightarrow \Psi$ realizable

- $\Psi$ realizable $\Rightarrow$
  $\exists$ a computation tree $\in \mathcal{L}(A_{\Psi,\Psi}) \Rightarrow$
  $\color{red}{x-ray(wide}_{(2^E)}(<(2^I)^*,P>)) \in \mathcal{L}(A_{\Psi,\Psi}) \Rightarrow$
  $\color{red}{wide}_{(2^E)}(<(2^I)^*,P>) \in \mathcal{L}(\text{cover}(A_{\Psi,\Psi})) \Rightarrow$
  $<\(2^I)^*,P> \in \mathcal{L}(\text{narrow}_{(2^E)}(\text{cover}(A_{\Psi,\Psi})))$

- Constructive emptiness check
Formally...
Formal Discussion

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

- $O((mn)^{3n})$ for nondeterministic tree automata
  - $m = |A|$  
  - $n =$ number of pairs in Rabin acceptance condition $\Gamma$

- Works by model checking (sort of)
  - $A\Phi_{\Gamma} = A(...) (GF(g_\gamma) \ FG(\neg b_\gamma)) ...$
    - $i \in \{0 \ldots m - 1\}$
“Model checking” Overview

• Nondeterministic tree automata aren't convenient

• Convert to an AND/OR diagram $T$, pseudo-model-check
  - $|T|$ is $O(|A|)$

• $\mu Y. \forall \gamma \in \Gamma \text{ AFAG}((\neg b_{\gamma} Y) \land A(Fg_{\gamma} \Phi_{\Gamma/\{\gamma\}}))$
  - $\Phi_{\Gamma}$ also adjusted
  - $T,s \models A\Phi_{\Gamma} \Leftrightarrow T,s \models \exists i.Y^i, i \leq |T|$
“Model checking” $A(Fg_\gamma \Phi_{\Gamma/\{\gamma\}})$

- LHS: $T,s$ $AFq \Leftrightarrow T,s$ $\mu x.(q \text{ EXAX}x)$
- $val(T, AFg_\gamma) \phantom{\text{\text{\text{\text{\text{}}}}} val(T, A\Phi_{\Gamma/\{\gamma\}})}$ doesn't work
  - Make it disjoint manually?
- RHS: $val(T/val(T, AFg_\gamma), A\Phi_{\Gamma/\{\gamma\}})$
  - Recursive in other pairs
“Model checking” $\text{AG}(g_\gamma(Y))$

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

- $\mu x. (\text{val}(T, \text{AG}(g_{\gamma}(Y)))) \text{ EXAX} x$
  - $T, s \quad \text{AFAG} g_{\gamma}(Y_i) \Rightarrow$
    - $s \in \text{val}(T, \text{AFAG} g_{\gamma}(Y_i)) \Rightarrow$
    - $T, s \quad \exists \alpha. \text{AFAG} g_{\gamma}(Y^\alpha)$

- Do all of that for each $\gamma$
  - $O(|\Gamma||T|^2)$ 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 $\psi$!
  - 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 $\mathcal{L}_{\text{spec}}$

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

- $A_\psi$ over $2^I$ $\circ$-labeled $2^I$-trees
  - $O(|\psi|)$ states
    - $q_0$, $\{ cl(\psi) \times \{ \exists, \forall \} \}$
  - Büchi acceptance condition
    - $A_\psi$ accepts $<T,V> \Leftrightarrow \text{wide'}_{2^E}(<T,V>)$ $\psi$
  - Transition function $\delta$ as in Yoram and Shahar's presentation
    - Adjusted for modes
The pre-transition relation $\delta'$

- $\delta'(\psi, \sigma) : cl(\psi) \times 2^I \xrightarrow{E \ O} B^+(2^I \times Q)$
  - $\delta'(p \in I, E \notin \emptyset, \sigma) = p \sigma$
  - Logical connectives carry through
  - $MX$ becomes $(, M)$
    - over inputs for $E$, over inputs for $A$
  - $M[1 \cup 2]$ becomes the usual or/recursive and
    - Same input connectives as $X$
  - $MG$ is the same as $M[true \cup ]$
The transition relation $\delta$

- For $\in cl(\psi)$ and $u \in 2^I$
  
  $\delta(<\, M>, u) = \delta'(\ , u \quad \tau)$

  - $M = \exists$: over all $\tau$ in $2^E$
  
  - $M = \forall$: over all $\tau$ in $2^E$

- $\delta(q_0, u) = \delta'(\psi, u)$

- Some reductions:
  
  - $p \in E$ can be reduced to true/false for $\exists/\forall$
  
  - $EX$ and $AX$ are $\delta'$ regardless of state mode

  - $\delta(<EX \ , M>, u) = \delta'(EX \ , u)$
2\textsuperscript{l}-exhaustiveness

- Recall: $A_\psi$ accepts $<T,V> \iff \text{wide'}_2 E(<T,V>) \psi$
- $V$ need not be $2^l$-exhaustive
  - i.e. $V(w) \neq 2^l \ dir(w) = 2^l$
  - $A_\psi$ 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 $A_\psi$ (alternating)
- Cross with $A_{\text{exh}}$ (nondeterministic)
- Perform constructive emptiness test
- EXPTIME
  - Exponential blowup on conversion from alternating to nondeterministic
  - Polynomial emptiness check
Discussion

• Useful?
• Tractable?
• Implementation?