Reliable and Interpretable Artificial Intelligence

Dana Drachsler-Cohen
ETH Zurich

Fall 2017
So far: CEGIS + Constraint-based Synthesis

Sketch defines the **hypothesis space** based on a function with “holes”

Synthesizer based on CEGIS: **queries oracle** to check candidate functions

**CEGIS** = **Counter Example Guided Inductive Synthesis**

Sketch:

```cpp
bit[W] isolate0Sketched(bit[W] x){
  return ~(x + ??) & (x + ??);
}
```

CEGIS:

```cpp
bit[W] isolate0Fast (bit[W] x) return ~x & (x+1);
```
Today (Part I): Synthesis from Examples

An example synthesizer: Microsoft’s Flash Fill

PBE synthesizers are **practical** but may not generalize to unseen data
Today (Part II): Exact Learning from Examples

Guaranteed to generalize on unseen data via supervision-guided synthesis

Result based on a 2017 paper, considered an open problem (yet, several open problems left)

An example synthesizer (Homework): learning time-series patterns

Exact PBE Synthesizer

A set of predicates $Q +$
Programming by Example (PBE) Synthesizer

Writing specifications is complex
• Error-prone for programmers
• Impossible for end users

Question: can a computer learn a function described by examples?
• Intuitive to convey user intent
• Requires no prior programming knowledge from the end user

Problem statement: Given a DSL and a fixed set of input-output examples, learn a function over the DSL that is consistent with the provided examples
Programming by Example – Historical Background

From the mid-70’s
  • Learn a program from input-output examples (Hardy, 1974)/concrete executions (Smith, 1975)

Very basic approaches
  • Types-based pruning, heuristic-guided search

As processors became better and constraint-solvers improved, practical PBE became feasible, e.g., Microsoft’s Flash Fill
Microsoft’s Flash Fill

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(425)-706-7709</td>
<td>425-706-7709</td>
</tr>
<tr>
<td>510.220.5586</td>
<td>510-220-5586</td>
</tr>
<tr>
<td>1 425 235 7654</td>
<td>425-235-7654</td>
</tr>
<tr>
<td>425 745-8139</td>
<td>425-745-8139</td>
</tr>
</tbody>
</table>

**Wired Magazine:**
Excel is now a lot easier for people who aren’t spreadsheet- and chart-making pros. The application’s new Flash Fill feature recognizes patterns, and will offer auto-complete options for your data. For example, if you have a column of first names and a column of last names, and want to create a new column of initials, you’ll only need to type in the first few boxes before Excel recognizes what you’re doing and lets you press Enter to complete the rest of the column.
Flash Fill’s Problem Definition

Given a set of input-output examples, exemplifying a desired string manipulation task, synthesize the desired string manipulation function

Flash Fill returns a function consistent with the given examples

Does not guarantee function generalizes on unseen data from a small set of examples chosen by the user
Flash Fill’s DSL (defines hypothesis space)

Hypothesis space is all switch-case functions defined by this Domain Specific Language (DSL):

String expr $P$ := $\text{Switch}((b_1, e_1), \ldots, (b_n, e_n))$

$\text{Bool} \ b$ := $d_1 \lor \ldots \lor d_n$

$\text{Conjunct} \ d$ := $\pi_1 \land \ldots \land \pi_n$

$\text{Predicate} \ \pi$ := $\text{Match}(v_i, r, k) \mid \neg \text{Match}(v_i, r, k)$

Trace expr $e$ := $\text{Concatenate}(f_1, \ldots, f_n)$

Atomic expr $f$ := $\text{SubStr}(v_i, p_1, p_2)$

$\mid \text{ConstStr}(s)$

$\mid \text{Loop}(\lambda w : e)$

Position $p$ := $\text{CPos}(k) \mid \text{Pos}(r_1, r_2, c)$

Integer expr $c$ := $k \mid k_1 w + k_2$

Regular Expression $r$ := $\text{TokenSeq}(T_1, \ldots, T_m)$

Token $T$ := $C + \mid [\neg C] +$

$\mid \text{SpecialToken}$
Flash Fill’s DSL (defines hypothesis space)

Hypothesis space is all switch-case functions defined by this Domain Specific Language (DSL):

Example:
\[\text{SubStr}(v_1, \text{Pos}(_\varepsilon, \text{NumTok}, 1), \text{CPos}(-1))\]

<table>
<thead>
<tr>
<th>Input (v_1)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTR KRNL WK CORN 15Z</td>
<td>15Z</td>
</tr>
<tr>
<td>CAMP DRY DBL NDL 3.6 OZ</td>
<td>3.6 OZ</td>
</tr>
<tr>
<td>CHORE BOY HD SC SPNG 1 PK</td>
<td>1 PK</td>
</tr>
<tr>
<td>FRENCH WORCESTERSHIRE 5 Z</td>
<td>5 Z</td>
</tr>
<tr>
<td>O F TOMATO PASTE 6 OZ</td>
<td>6 OZ</td>
</tr>
</tbody>
</table>
Searching the Hypothesis Space

**Ambiguity:** for every set of input-output examples, *many expressions* can explain how the output was generated.

Example: how to explain the following transformation?

\[
\text{\{Write “4”/copy the second digit\}} \times \\
\text{\{Write “2”/copy the third digit\}} \times \\
\text{\{Write “5”/copy the fourth digit\}} \times \\
\text{\{Write “–”/copy the fifth digit/copy the sixth digit\}} \times \ldots
\]
Flash Fill’s Hypothesis Space

To efficiently represent the space, define a DAG:

- Nodes are expressions over substring
- A path describes the (concatenated) output

Easy to intersect with additional examples

- Even after intersecting, many expressions remain
- Eventually, Flash Fill picks one heuristically

• Data structure key to Flash Fill success
• What about conditionals and loops? Heuristics

(425)-706-7709

425-706-7709

510.220.5586

510-220-5586

\begin{itemize}
  \item Copy 2\textsuperscript{nd} char
  \item Copy 3\textsuperscript{rd} char
  \item Copy 4\textsuperscript{th} char
  \item Write “4”
  \item Write “2”
  \item Write “5”
\end{itemize}
PBE: Fundamental Challenge in Learning

User provides examples, PBE synthesizer finds a function satisfying these

Function **may not generalize** to unseen data (examples)
High-level View

**Goal**: learn a function from examples that generalizes to unseen data

**Approach**: split synthesis/learning task in two steps:
- Step 1: learn the user intent, modeled as a formula in some hypothesis space
- Step 2: given the formula, synthesize the low-level function (e.g., like Sketch)

To learn user intent, we phrase this problem as **exact learning**. Exact learning deals with learning concepts through examples, without posing too many queries
Exact Learning (by Angluin)

• Study the problem of learning a target hypothesis
  • Given a domain $D$, a hypothesis is a subset $\overline{H} \subseteq D$ or a formula
• Instead of a set of examples there is a source that can be queried
  • Source is known as the teacher or oracle
• Goal: learn the target $\overline{H}$ while minimizing the number of queries
• Two types of questions:
  • Membership queries: Is $e \in \overline{H}$?
  • Equivalence queries: Is $H=\overline{H}$?
Exact Learning: Problem Definition

Given:

- a domain $D$ (possibly infinite),
- a target function $\bar{P}$
- an oracle (user) that can answer whether $\bar{P}(in) = out$,

Learn a function $P$, such that for all $in \in D$: $P(in) = \bar{P}(in)$ while minimizing the number of queries.

Compared to PBE:
- No initial set of examples $E$
- PBE guarantee:
  - For $e \in E$: $P(in) = \bar{P}(in)$
Current PBE that are Exact(-ish)

Exhaustive search - Oracle-guided (Jha et al. 2010)
   While \( \exists \bar{P}, \bar{P}', \text{in: } P(\text{in}) \neq \bar{P}'(\text{in}) \)
      Ask for \( \bar{P}(\text{in}) \)
      Prune all \( P: P(\text{in}) \neq \bar{P}(\text{in}) \)

Using equivalence queries - CEGIS (Solar-Lezama, 2008)
   \( P = \text{Learn}(E) \)
   Ask \( P = \bar{P} \)? Yes – complete, no - \( E \cup \{\text{ask counterexample()}\} \)

Unguided searches, may pose too many queries (as we will see)
Idea: Implication Graphs

Identify implications between nodes in the hypothesis space

Define a directed graph and search wisely:
- If a node is okay: target hypothesis is reachable
- If a node is not okay: target hypothesis not reachable

Enables substantial pruning
Two-step Synthesis

1. Learn (complete) specifications (formulas), capturing user intent on all possible inputs. Works by phrasing problem in exact learning.

2. Synthesize function: phrase as a problem of synthesis from (complete) specifications.

What if we want the search to be hypothesis-directed?
Exact Learning of Specifications: Problem Definition

Given:

- A domain $D$ and a set of predicates $S$ (predicate is $R \subseteq D^n$)
- A target function $\bar{P}$ that meets a specification $\psi$ over $S$
  - $\psi$ is assumed to be disjunctive
- An oracle (user) that can answer whether $\bar{P}(in) = out$

learn a formula $\varphi$, such that: $\varphi \equiv \psi$ while minimizing the number of queries

Based on “Learning Disjunctions of Predicates”, N. Bshouty, D. Drachsler-Cohen, M. Vechev, E. Yahav, COLT, 2017
Problem Definition via Example

Efteling is an amusement park in the Netherlands. They keep a list of all visitors’ name, gender and age from which they infer properties.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td>Gender</td>
<td>Age</td>
<td>isMale</td>
<td>isFemale</td>
<td>isAdult</td>
<td>isMan</td>
<td>isWoman</td>
<td>isChild</td>
<td>isBoy</td>
<td>isGirl</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Gender = M</td>
<td>Gender = F</td>
<td>Age &gt; 18</td>
<td>isMale &amp; is Adult</td>
<td>isFemale &amp; isAdult</td>
<td>Age &lt;= 18</td>
<td>isMale &amp; isChild</td>
<td>isFemale &amp; isChild</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sophia</td>
<td>F</td>
<td>20</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>Alden</td>
<td>M</td>
<td>12</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>5</td>
<td>Emma</td>
<td>F</td>
<td>16</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>6</td>
<td>Jackson</td>
<td>M</td>
<td>30</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>Olivia</td>
<td>F</td>
<td>15</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>Ethan</td>
<td>M</td>
<td>2</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>9</td>
<td>Liam</td>
<td>M</td>
<td>60</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>10</td>
<td>Isabella</td>
<td>F</td>
<td>10</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Problem Definition via Example

Efteling opens a new roller coaster ride that has a height limit of at most 1.5 m. The marketing team wants to advertise this ride only to its child and female visitors:

\[ \varphi(e) = \text{isFemale}(e) \lor \text{isBoy}(e) \]

Unfortunately, they do not know to express this specification. Can they convey their intent to a synthesizer only through examples?
Problem Definition via Example

Exact PBE

- (Olivia, F, 15) True
- (Jackson, M, 30) False
Formal Problem Definition

Let $Q$ be a finite set of Boolean functions over a domain $D$ (possibly infinite).

- $D = \{(Sophia,F,15),(Aiden,M,12), \ldots \}$
- $Q = \{isMale, isFemale, \ldots \}$
Formal Problem Definition

Let $Q$ be a finite set of Boolean functions over a domain $D$ (possibly infinite). Given an oracle that has a target function $\psi \in Q_V = \{ \vee q \in P \mid P \subseteq Q \}$, who can answer membership queries: given $e \in D$, she returns $\psi(e)$.

- $D = \{(Sophia,F,15),(Aiden,M,12), \ldots \}$, $Q = \{isMale, isFemale, \ldots \}$
- $\varphi = isFemale \lor isBoy$
- $\varphi(Sophia,F,15) = 1$, $\varphi(Aiden,M,12) = 1$, $\varphi(Jackson,M,30) = 0$
Learning Disjunctions: Problem Definition

Let $Q$ be a finite set of Boolean functions over a domain $D$ (possibly infinite). Given an oracle that has a target function $\psi \in Q_{\lor} = \{ \lor_{q \in P} q \mid P \subseteq Q \}$, that can answer membership queries: given $e \in D$, she returns $\psi(e)$. The goal of the learner is to find $\psi$ with a minimum number of queries.

We will often call Boolean functions predicates. We call formulas of the form $\lor_{q \in P} q$ disjunctive formulas.

We show a synthesizer that poses at most $|Q| \cdot OPT(Q_{\lor})$ membership queries, where $OPT(Q_{\lor})$ is the worst case number of membership queries.
Hypothesis Space

As with Flash Fill, the crux is to represent the candidate solutions in a hypothesis space that enables efficient pruning.

Main idea:

- Define the set of nodes to be a set of non-equivalent formulas.
- The edges will convey the implications between the nodes.
- Pruning a node will imply its descendants can be pruned as well.
The Nodes of the Hypothesis Space

The nodes $N$ are a subset of $Q_\lor$ such that for all $\psi \in Q_\lor$ there exists $\psi' \in N$, such that $\psi \equiv \psi'$.

To define $N$, we use the equivalence relation $\equiv$ over formulas in $Q_\lor$

\[
\text{isFemale } \lor \text{isBoy } \equiv \text{isFemale } \lor \text{isChild}
\]

If $\psi_1 \equiv \psi_2$, then $\psi_1 \lor \psi_2 \equiv \psi_1 \equiv \psi_2$. In particular,

\[
[\psi_1 \lor \psi_2] = [\psi_1]
\]

\[
\text{isFemale } \lor \text{isBoy } \equiv \text{isFemale } \lor \text{isBoy } \lor \text{isChild}
\]

The representatives of the equivalence classes are the formulas whose extensions with other predicates are non-equivalent to them

\[
[\text{isFemale } \lor \text{isBoy}] \rightarrow \text{isFemale } \lor \text{isBoy } \lor \text{isChild } \lor \text{isGirl } \lor \text{isWoman}
\]
The Edges of the Hypothesis Space

We define a partial order over $N$, denoted $\Rightarrow$
Then, the hypothesis space is the Hasse diagram over $\Rightarrow$
That is, the ordering is by logical implication

$\varphi_1 \Rightarrow \varphi_2$ if $\varphi_1$ logically implies $\varphi_2$
That is, if $e \in D$ such that $\varphi_1(e) = 1$, then $\varphi_2(e) = 1$

Example:

$\text{isWoman} \Rightarrow \text{isFemale} \lor \text{isWoman} \lor \text{isGirl}$
The Hasse Diagram

\[ \text{isMale} \lor \text{isFemale} \lor \text{isAdult} \lor \text{isMan} \lor \text{isWoman} \lor \text{isChild} \lor \text{isBoy} \lor \text{isGirl} \]

\[ \text{isMale} \lor \text{isAdult} \lor \text{isMan} \lor \text{isWoman} \lor \text{isBoy} \]

\[ \text{isFemale} \lor \text{isAdult} \lor \text{isMan} \lor \text{isWoman} \lor \text{isGirl} \]

\[ \text{isMan} \lor \text{isMan} \lor \text{isBoy} \]

\[ \text{isFemale} \lor \text{isWoman} \lor \text{isGirl} \]

\[ \text{isChild} \lor \text{isBoy} \lor \text{isGirl} \]

\[ \text{isAdult} \lor \text{isMan} \lor \text{isWoman} \]

\[
\text{isMale} \lor \text{isMale} \lor \text{isMan} \lor \text{isMan} \lor \text{isBoy} \]

\[
\text{isFemale} \lor \text{isFemale} \lor \text{isWoman} \lor \text{isWoman} \lor \text{isGirl} \]

\[
\text{isChild} \lor \text{isBoy} \lor \text{isGirl} \]

\[
\text{isAdult} \lor \text{isMan} \lor \text{isWoman} \]

The maximal element (top)

The minimal element (bottom)
Explore Hasse Diagram Top-Down
(so we introduce direction)

\[ \text{isMale } \lor \text{isFemale } \lor \text{isAdult } \lor \text{isMan } \lor \text{isWoman } \lor \text{isChild } \lor \text{isBoy } \lor \text{isGirl} \]

The maximal element (top)

The minimal element (bottom)
Properties of the Hypothesis Space

Denote $Q(\varphi)$ to be the set of predicates from $Q$ in $\varphi$.

$$Q(isChild \lor isBoy \lor isGirl) = \{isChild, isBoy, isGirl\}$$

Lemma 1: If $\varphi_1$ is a descendant of $\varphi_2$, then $Q(\varphi_1) \subseteq Q(\varphi_2)$
The GCD of Nodes

The greatest common descendant (GCD) of $\varphi_1, \varphi_2$ is the maximal common descendant in the Hasse diagram.

Lemma 2: $Q(\gcd(\varphi_1, \varphi_2)) = Q(\varphi_1) \cap Q(\varphi_2)$

In particular, $\lor (Q(\varphi_1) \cap Q(\varphi_2)) \in N$

Note this property does not hold for the lowest common ancestor.
Witnesses

A witness for $\varphi_1, \varphi_2 \in N$ is an example $e \in D$ such that $\varphi_1(e) \neq \varphi_2(e)$

Lemma 3: Let $\varphi_1$ be a child of $\varphi_2$ and $e$ a witness for them. Then:

- $\varphi_1(e) = 0, \varphi_2(e) = 1$
  
  $\text{isMale} \lor \text{isAdult} \lor \text{isMan} \lor \text{isBoy}(\text{Sophia}, \text{F}, 20) = 0$
  $\text{isMale} \lor \text{isAdult} \lor \text{isMan} \lor \text{isWoman} \lor \text{isBoy}(\text{Sophia}, \text{F}, 20) = 1$

- For every $q \in Q(\varphi_1), q(e) = 0$
  
  $\text{isMale}(\text{Sophia}, \text{F}, 20) = 0, \text{isMan}(\text{Sophia}, \text{F}, 20) = 0, \text{isBoy}(\text{Sophia}, \text{F}, 20) = 0$

- For every $q \in Q(\varphi_2) \setminus Q(\varphi_1), q(e) = 1$
  
  $\text{isAdult}(\text{Sophia}, \text{F}, 20) = 1, \text{isWoman}(\text{Sophia}, \text{F}, 20) = 1$
Witnesses are Unique

**Lemma 4:** Let \( \{ \varphi_1, ..., \varphi_k \} \) be the children of \( \varphi \). If \( e_i \) is a witness for \( \varphi_i \) and \( \varphi \), then \( e_i \) is not a witness for \( \varphi, \varphi_j \) for any \( j \neq i \).
Witnesses Characterize the Hypothesis Space

**Lemma 5:** Let $\varphi_i$ be a child of $\varphi$, $e_i$ a witness for them, and $\varphi_j$ a descendant of $\varphi$

- If $\varphi_j(e_i) = 0$, $\varphi_j$ is a descendant of $\varphi_i$ or equal to $\varphi_i$
- If $\varphi_j(e_i) = 1$, $\varphi_j$ is not a descendant of $\varphi_i$ nor equal to $\varphi_i$

```
isMale ∨ isFemale ∨ isAdult ∨ isMan ∨ isWoman ∨ isChild ∨ isBoy ∨ isGirl
```

Olivia, F, 15

```
isMale ∨ isAdult ∨ isMan ∨ isWoman ∨ isBoy
```

```
isFemale ∨ isAdult ∨ isMan ∨ isWoman ∨ isGirl
```

```
isMale ∨ isMan ∨ isBoy
```

```
isFemale ∨ isWoman ∨ isGirl
```

```
isChild ∨ isBoy ∨ isGirl
```

```
isAdult ∨ isMan ∨ isWoman
```

```
isMan
```

```
isWoman
```

```
isBoy
```

```
isGirl
```
The D-SPEX Synthesizer

We are finally ready to define our synthesizer that learns **Disjunctive Specifications from Examples**

Main idea:

- Start from the top element and traverse downwards
- At each step, consider one node $\varphi$ and generate witnesses for all its children
- Present a membership query for each witness
- If the oracle classified a witness $e_i$ for $\varphi, \varphi_i$ as a positive example $\psi(e_i) = 1$: the target formula is not $\varphi_i$ or its descendants (Lemma 5)
- Otherwise, $\psi(e_i) = 0$: the target formula is $\varphi_i$ or its descendants (Lemma 5)
- Execute recursively on the GCD of all “accepted” children
The D-SPEX Synthesizer

\[
\text{D-SPEX}(\varphi, T): \quad \text{// initially, call with D-SPEX}(\bigvee_{q \in Q} q, \emptyset)
\]

\[
Q \leftarrow Q(\varphi);
\]

\[
\text{Flag} \leftarrow 1
\]

\[
\text{for } \varphi' \in \text{children}(\varphi):
\]

\[
\text{if } (\forall R \in T. Q(\varphi') \not\subset R)
\]

\[
e \leftarrow \text{witness}(\varphi, \varphi')
\]

\[
\text{if } (\text{mem}(e) = 0)
\]

\[
Q = Q \cap Q(\varphi'); \text{Flag} \leftarrow 0
\]

\[
\text{else } T \leftarrow T \cup \{Q(\varphi')\}
\]

\[
\text{if } (\text{Flag} = 1) \quad \text{return } \bigvee_{q \in Q} q
\]

\[
\text{D-SPEX}(\bigvee_{q \in Q} q, T')
\]
Running Example

T:

\[
isMale \lor isAdult \lor isMan \lor isWoman
\lor isBoy
\lor isFemale \lor isAdult \lor isMan \lor isWoman
\lor isGirl
\lor isMale \lor isMan \lor isChild \lor isBoy \lor isGirl
\]

D-SPEX(φ,T):

\[
Q \leftarrow Q(\phi); \ Flag \leftarrow 1
\]

For \( \phi' \in \text{children}(\phi) \):

\[
\text{If } (\forall R \in T. Q(\phi') \not\in R)
\]

\[
e \leftarrow \text{witness}(\phi, \phi')
\]

\[
\text{If } (\text{mem}(e) = 0) Q = Q \cap Q(\phi'); \ Flag \leftarrow 0
\]

\[
\text{Else } T \leftarrow T \cup \{Q(\phi')\}
\]

\[
\text{If } (\text{Flag} = 1) \text{ return } V_{q \in Q} q
\]

\[
\text{D-SPEX}(V_{q \in Q} q, T)
\]
Running Example

T: 
- isMale ∨ isAdult ∨ isMan ∨ isWoman 
- isFemale ∨ isAdult ∨ isMan ∨ isWoman 
- isMale ∨ isMan ∨ isChild ∨ isBoy ∨ isGirl

D-SPEX(\(\varphi, T\)):

- \(Q \leftarrow Q(\varphi); Flag \leftarrow 1\)
- For \(\varphi' \in \text{children}(\varphi)\):
  - If (\(\forall R \in T. Q(\varphi') \notin R\))
    - \(e \leftarrow \text{witness}(\varphi, \varphi')\)
    - If (\(\text{mem}(e) = 0\)) \(Q = Q \cap Q(\varphi'); Flag \leftarrow 0\)
    - Else \(T \leftarrow T \cup \{Q(\varphi')\}\)
  - If (\(Flag = 1\)) return \(\forall q \in Q \ q\)
- D-SPEX(\(\bigvee_{q \in Q} q, T\))
Complexity Analysis

Theorem: If the children and witnesses of any $\varphi \in N$ can be found in time $t$, then D-SPEX learns $\psi$ in time $t \cdot |Q|$ and $|Q| \cdot \max_{n \in N}(\text{children}(n))$ membership queries.
Lower Bound

**Theorem:** any learning algorithm that learns $Q_v$ must pose at least $\max(\log(|N|), \max_{n \in N} |\text{Children}(n)|)$. In particular, D-SPEX poses at most $|Q| \cdot \text{OPT}(Q_v)$.

Let $n$ be a node with a maximal number of children. If the target is $n$, any learning algorithm must pose for every child a membership query that satisfies $n$ but not this child

- The membership queries are unique for every child

At every invocation of D-SPEX, the number of membership queries is bounded by the maximal number of children (i.e., $\text{OPT}(Q_v)$)

The number of invocations is bounded by $|Q|$. 
Missing Points

How to obtain a node’s children?

How to efficiently compute the witnesses?
PBE Synthesis: Key Points

Efficiently represent the hypothesis space
• Identify equivalent formulas and define nodes only for representatives
• Express possible pruning through edges

Identify critical examples (witnesses)

Design a synthesizer that traverses a polynomial number of nodes
Open Research Questions

What can we prove about learning the class of DNFs?
  • In general, has to be exponential

Can we improve the lower bound? Can we reduce the number of membership queries D-SPEX asks?

Can we improve the number of queries D-SPEX asks in specific application domains?
Summary

• In lecture 2, we saw constraint-based synthesis through CEGIS

• Today, we saw another kind of supervised synthesis where the goal was to synthesize a formula capturing the user intent

• We showed a (restricted) hypothesis space in which we could generalize to unseen data

• Open problems concern other hypothesis spaces and predicate sets
**HW: An End-to-End Exact PBE**

**Technical Analysis:**
Predict price direction using current prices

**Patterns:**
Special forms that signal whether to buy or sell

**Goal:** Synthesize a function (a model) detecting a pattern

\[
P_0 = \text{LLV}(\text{Close}, W);
BP_0 = \text{LLVBars}(\text{Close}, W);
P_1 = \text{HHV}(\text{Close}, BP_0);
BP_1 = \text{HHVBars}(\text{Close}, BP_0);
P_2 = \text{LLV}(\text{Close}, BP_1);
BP_2 = \text{LLVBars}(\text{Close}, BP_1);
P_3 = \text{HHV}(\text{Close}, BP_2);
BP_3 = \text{HHVBars}(\text{Close}, BP_2);
P_4 = \text{LLV}(\text{Close}, BP_3);
BP_4 = \text{LLVBars}(\text{Close}, BP_3);
P_5 = \text{HHV}(\text{Close}, BP_4);
BP_5 = \text{HHVBars}(\text{Close}, BP_4);
P_6 = \text{LLV}(\text{Close}, BP_5);
\]

Filter: \( P_0 < P_1 \) AND \( P_2 < P_1 \) AND \( P_1 < P_3 \) AND \( P_5 < P_3 \) AND \( P_4 < P_5 \) AND \( P_6 < P_5 \);