Introduction

- Inductive Synthesis of Recursive Functional Programs [1, 4, 5]
  - challenging subfield of machine learning
  - still little researched niche
- Automated Program Construction
  - from incomplete specifications (I/O examples)
    \[
    \begin{align*}
    [A] & \to A \\
    [A, B] & \to B \\
    [A, B, C] & \to C
    \end{align*}
    \]
  - Last([X]) → X
  - Last([X|X]) → Last([X])
- Potential Applications
  - end-user programming
  - assist professional programmers (Systems Engineering)
  - automatically invent new and efficient algorithms

Description of Problems

1. Single Recursive Call without Predicate Invention

   \( \text{evenpos}(X, Y) \) holds if list \( Y \) contains all elements of list \( X \) at an even position in unchanged order.

   \( \text{insert}(X, Y, Z) \) holds if \( X \) is a list with its elements in a not decreasing order, and \( Z \) is \( X \) with \( Y \) inserted on the right place.

   \( \text{inslast}(X, Y, Z) \) holds if \( Z \) is the last list with \( Y \) inserted at the end.

   \( \text{last}(X, Y) \) holds if \( Y \) is the last element of the list \( X \).

   \( \text{length}(X, Y) \) holds if \( Y \) is the length of the list \( X \).

   \( \text{member}(X, Y) \) holds if \( X \) is a list containing the element \( Y \).

   \( \text{switch}(X, Y) \) holds if \( Y \) can be obtained from list \( X \) all elements on an odd position changed place with their right neighbour.

   \( \text{unpack}(X, Y) \) holds if \( Y \) is a list of lists, each containing one element of \( X \) in unchanged order.

2. Single Recursive Call with Predicate Invention

   \( \text{i-sort}(X, Y) \) holds if \( Y \) is a permutation of list \( X \) with elements in a non-decreasing order.

   \( \text{nullall}(X, Y) \) holds if \( Y \) contains nothing but the last element of list \( X \) as many times as the number of elements in \( X \).

   \( \text{reverse}(X, Y) \) holds if \( Y \) is the reverse of list \( X \).

   \( \text{shift}(X, Y) \) holds if \( Y \) could be derived from list \( X \) by shifting the first element to the end.

   \( \text{swap}(X, Y) \) holds if \( Y \) could be derived from list \( X \) by swapping the first and the last element.

3. Multiple Recursive Call with(out) Predicate Invention

   \( \text{lasts}(X, Y) \) holds if \( Y \) is a list of lists, and \( Y \) contains the last elements of each list in \( X \) in correct order.

   \( \text{flatten}(X, Y) \) holds if \( Y \) is the flattened version of the list of lists \( X \).

4. Miscellaneous Problems

   \( \text{mergelists}(X, Y, Z) \) holds if list \( Z \) could be derived from the lists \( X \) and \( Y \) such that \( Z = [x_1, y_1, x_2, y_2, \ldots] \) where each \( x_i \) and \( y_i \) is the \( i \)th element of the list \( X \) and \( Y \), respectively.

   \( \text{odd}(X) \text{/} \text{even}(X) \) holds if \( X \) is an odd, respectively even number, and each predicate is defined in terms of \( \text{zero}(X) \) and the other.

The Systems

- Adate [6]
  - (Automatic Design of Algorithms Through Evolution)
  - utilising evolutionary computation
  - induces functional programs in a subset of ML
  - user provided initial program is evolved
- Atre [2]
  - (Approcciomando di Teoriche Risonano da Esecuzione)
  - search space are definite clauses
  - general-to-specific parallel beam search
  - specialized to learning multiple recursive concepts
- Dialogs-II [3]
  - (Dialog-based Inductive and Abductive LOGic program Synthesiser)
    - inductive and abductive
    - schema-guided
    - queries interactively for evidence

Problem Classes

I. Single recursive call, no predicate invention

   - solvable with a single recursive call in the body of the predicate definition; no predicate or variable invention is required.

II. Single recursive call with predicate invention

   - at least the invention of an auxiliary predicate is required.

III. Multiple recursive call

   - at least a second recursive call is necessary (either of an other recursive predicate or of the target predicate itself)

V. Miscellaneous

   - emphasises the individual strengths of a certain system

Conclusion

- combine Dialogs-II’s search bias with Adate’s unrestricted search space
- exploit expressional power of functional languages
- adopt Atre’s k-beam search strategy
- learn mutually dependent recursive target functions
- our system Igor [5] formalises functional program synthesis in the term-rewriting framework
- functional programs as constructor term rewriting systems

Results of the Test Setting

<table>
<thead>
<tr>
<th></th>
<th>(1.)</th>
<th>(2.)</th>
<th>(3.)</th>
<th>(4.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adate</td>
<td>2.0</td>
<td>1.5</td>
<td>1.2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>2.7</td>
<td>2.8</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>78</td>
<td>70</td>
<td>232</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>43</td>
<td>110</td>
<td>822</td>
<td>0.05</td>
</tr>
<tr>
<td>Adate</td>
<td>91.6</td>
<td>17.9</td>
<td>6.4</td>
<td>1983</td>
</tr>
<tr>
<td></td>
<td>0.19</td>
<td>0.03</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>1.9</td>
<td>0.06</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>0.15</td>
<td>0.11</td>
<td>0.13</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td></td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.05</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>0.19</td>
<td>0.06</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>0.19</td>
<td>0.10</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>0.05</td>
<td>0.05</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
<td>0.19</td>
<td>0.06</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>0.19</td>
<td>0.10</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

References