Combining Analytical and Evolutionary Inductive Programming

Neil Crossley Emanuel Kitzelmann Martin Hofmann Ute Schmid

Cognitive Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

The Second Conference on Artificial General Intelligence, 2009

Inductive Programming (IP)

Automatic construction of recursive (functional) programs from input/output-examples or other kinds of incomplete specifications.

Example

I/O-examples (containing variables):

$$Switch([\,]) = [\,], Switch([X]) = [X], Switch([X, Y]) = [Y, X],$$
$$Switch([X, Y, Z]) = [Y, X, Z], Switch([X, Y, Z, V]) = [Y, X, V, Z]$$

Induced program:

$$\begin{aligned} Switch([\,]) &= [\,] \\ Switch([X]) &= [X] \\ Switch([X, Y \mid Xs]) &= [Y, X \mid Switch(Xs)] \end{aligned}$$

Analytical vs. Evolutionary IP

Analytical IP

- **Detect syntactic regularities between I/O-examples** and derive recursive function definition from them.
- Assumes syntactically **restricted program schema**.
- **Fast**.

Evolutionary / Generate-and-Test IP

- **Enumerate program class**, use **I/O-examples for testing** only.
- Allows for fairly **unrestricted program classes**.
- **Expensive**.

Combining Analytical and Evolutionary IP

Try to combine the strengths of both approaches to overcome its weaknesses.

The approach: **Connect both methods in series**.

1. Efficiently generate an unfinished **program sketch** in a restricted program space by analytical induction.
2. Use this sketch as seed for an evolutionary search in an unrestricted program space.

Implementation

Tools

IGOR2 to analytically construct a program sketch.

ADATE **to evolve the final program based on Igor2's sketch.**

Necessary Implementation Steps

- A new stop criterion for IGOR2.
- An algorithm that builds a valid ADATE specification from an IGOR2 specification and a program sketch.

We have not yet implemented the method but hand-coded ADATE *specifications for our initial experiments.*

The ADATE Search

- Systematic (no randomization), global search.
- Several heuristics and program transformation operators, specialized to evolve recursive functional (SML) programs.
- Programs with low syntactic complexity and low time complexity are preferred.
- ADATE starts with a single individual, typically the empty program that only raises an exception.
- Then search goes towards bigger-and-bigger and better-and-better programs.

ADATE Specifications

Main Elements of an ADATE Specification

- Type- and (help-)function definitions.
- Declarations/definitions of target function `f` and main function (a context program that calls `f`) `main`, e.g.,

```
f ( <params> ) : <type> = raise D1
main ( <params> ) : <type> = f ( <params> )
```

- I/O-examples

  more generally: training inputs and an output evaluation function.

Example of a Sketch Developed by IGOR2

I/O-examples

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y]) &= [Y, X] \\
Switch([X, Y, Z]) &= [Y, X, Z] \\
Switch([X, Y, Z, V]) &= [Y, X, V, Z]
\end{aligned}
$$

Solution

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y \mid Xs]) &= [Y, X \mid Switch(Xs)]
\end{aligned}
$$

Example of a Sketch Developed by IGOR2

I/O-examples

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y]) &= [Y, X] \\
Switch([X, Y, Z]) &= [Y, X, Z] \\
Switch([X, Y, Z, V]) &= [Y, X, V, Z]
\end{aligned}
$$

Solution

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y \mid Xs]) &= [Y, X \mid Switch(Xs)]
\end{aligned}
$$

Example of a Sketch Developed by IGOR2

I/O-examples

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y]) &= [Y, X] \\
Switch([X, Y, Z]) &= [Y, X, Z] \\
Switch([X, Y, Z, V]) &= [Y, X, V, Z]
\end{aligned}
$$

Solution

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y \mid Xs]) &= [Y, X \mid Ys]
\end{aligned}
$$

Example of a Sketch Developed by IGOR2

I/O-examples

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y]) &= [Y, X] \\
Switch([X, Y, Z]) &= [Y, X, Z] \\
Switch([X, Y, Z, V]) &= [Y, X, V, Z]
\end{aligned}
$$

Solution

$$
\begin{aligned}
Switch([\,]) &= [\,] \\
Switch([X]) &= [X] \\
Switch([X, Y \mid Xs]) &= ?
\end{aligned}
$$

Unassisted

- The sketch is not used at all; ADATE is simply given the same examples as IGOR2 and must find the solution from scratch.
- Base line to test whether providing a seed really helps.

Example

```
f ( xs:list ) : list = raise D1
main ( xs:list ) : list = f ( xs )
```

Redefined

The target function `f` is **redefined** according to the sketch; ? is replaced by `raise D0`.
The Sketch may be revised by ADATE.

Example

```
fun f Xs =
  case Xs of
    nill => Xs
  | cons(Y1, Y2) => case Y2 of
                      nill => Xs
                    | cons(W1, W2) => raise D1

fun main Xs = f Xs
```

Restricted

The sketch is introduced as a predefined function g called by `main`; ? is replaced by a call to `f`.

The sketch is saved from being revised.

Example

```
fun f Xs = raise D1

fun g Xs =
  case Xs of
    nill => Xs
  | cons(Y1, Y2) => case Y2 of
                      nill => Xs
                    | cons(W1, W2) => f Xs

fun main Xs = g Xs
```

Functions

Any auxiliary functions inferred by IGOR2 may be included into the seed—either as predefined function to be called by `f` or as an inner function of `f` also subject to transformations.

Success

Problem	Type of Seed	Time	Efficiency
Switch	Unassisted	4.34	302
	Restricted	0.47	344
	Redefined	3.96	302
Sort	Unassisted	457.99	2487
	Restricted	225.13	2849
Swap	Unassisted	292.05	1076
	Restricted and functions	41.43	685
Lasts	Unassisted	260.34	987
	Restricted	6.25	1116
ShiftR	Unassisted	8.85	239
	Redefined, functions	1.79	239
ShiftL	Unassisted	4.17	221
	Restricted	0.61	281
Insert	Unassisted	7.81	176
	Restricted	18.37	240

Conclusion

- Analytical methods and generate-and-test based methods alone cannot solve relevant problems for different reasons.
- Idea: Combine both approaches.
- First experiments show, that the evolutionary system ADATE indeed can benefit from a partial solution (efficiently generated by IGOR2) as starting point.
- One problem is the different preference biases of IGOR2 and ADATE.