Inductive Synthesis of Recursive Functional Programs
A Comparison of Three Systems

Martin Hofmann, Andreas Hirschberger, Emanuel Kitzelmann, and Ute Schmid
Faculty of Information Systems and Applied Computer Science University of Bamberg
{martin.hofmann, andreas.hirschberger}@stud.uni-bamberg.de, {emanuel.kitzelmann, ute.schmid}@wiai.uni-bamberg.de

Introduction

- **Inductive Synthesis of Recursive Programs**[1, 4, 5]
  - **challenging subfield** of machine learning
  - still **little researched niche**
- **Automated Program Construction**
  - from incomplete specifications (**I/O examples**)

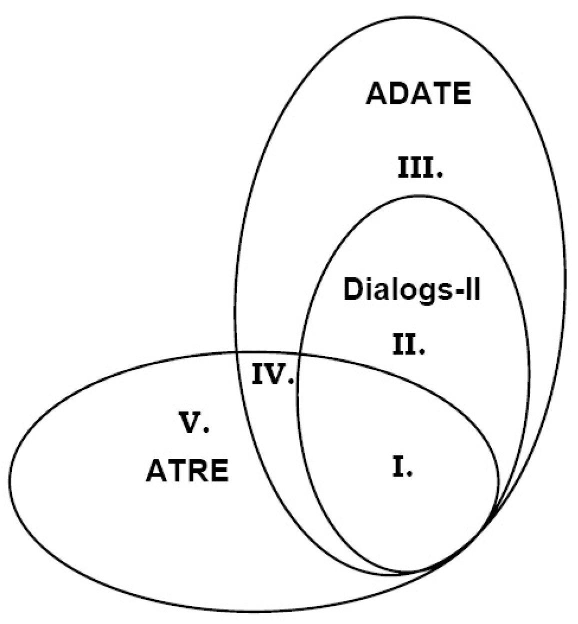
$$\left.\begin{array}{ll}\texttt{[A]} & \rightarrow \texttt{A}\\ \texttt{[A,B]} & \rightarrow \texttt{B}\\ \texttt{[A,B,C]} & \rightarrow \texttt{C}\end{array}\right\rangle \quad \begin{array}{ll}\texttt{Last([X])} & \rightarrow \texttt{X}\\ \texttt{Last([X|Xs])} & \rightarrow \texttt{Last([Xs])}\end{array}$$

- **Potential Applications**
  - **end-user programming**
  - assist professional programmers (**Systems Engineering**)
  - **automatically invent** new and efficient **algorithms**

The Systems

- **Adate** [6]
  (**A**utomatic **D**esign of **A**lgorithms **T**hrough **E**volution)
  - utilising **evolutionary computation**
  - induces **functional programs in** a subset of **ML**
  - user provided initial program is evolved
- **Atre** [2]
  (**A**pprendimento di **T**eorie **R**icorsive da **E**sempi)
  - search space are definite clauses
  - **general-to-specific parallel beam search**
  - specialized to learning **multiple recursive concepts**
- **Dialogs-II** [3]
  (**D**ialogue-based **I**nductive and **A**bductive **LOG**ic program **S**ynthesiser)
  - **inductive** and abductive
  - **schema-guided**
  - queries **interactive**ly 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. + IV. Multiple recursive call: at least a second recursive call is necessary (either of another recursive predicate or of the target predicate itself)

V. + III. Miscellaneous: emphasises the individual strengths of a certain system.

Classes III. and VI. were combined, since DIALOGS-II is not capable of multiple recursive calls and an ATRE specification for such a problem would result in an extensive enumeration of input/output pairs.

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**

Description of Problems

(1.) Single Recursive Call without Predicate Invention

evenpos(X, Y) holds iff list Y contains all elements of list X at an even position in unchanged order.

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

inslast(X, Y, Z) holds iff Z is the list X with Y inserted at the end.

last(X, Y) holds iff Y is the last element of the list X.

length(X, Y) holds iff Y is the length of the list X.

member(X, Y) holds iff X is a list containing the element Y.

switch(X, Y) holds iff list Y can be obtained from list X were all elements on an odd position changed place with their right neighbour.

unpack(X, Y) holds iff Y is a list of lists, each containing one element of X in unchanged order.

(2.) Single Recursive Call with Predicate Invention

i-sort(X, Y) holds iff the list Y is a permutation of list X with elements in a non decreasing order.

multlast(X, Y) holds iff the list Y contains nothing but the last element of list X as many times as the number of elements in X.

reverse(X, Y) holds iff the list Y is the reverse of list X.

shift(X, Y) holds iff list Y could be derived from list X by shifting the first element to the end.

swap(X, Y) holds iff list Y could be derived from list X by swapping the first and the last element.

(3.) Multiple Recursive Call with(out) Predicate Invention

lasts(X, Y) holds iff X is a list of lists, and Y contains the last elements of each list in X in the correct order.

flatten(X, Y) holds iff Y is the flattened version of the list of lists X.

(4.) Miscellaneous Problems

mergelists(X, Y, Z) holds iff the 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_n and y_n is the nth of the list X and Y, respectively.

odd(X)/even(X) holds iff X is an odd, respectively even number, and each predicate is defined in terms of $zero(X)$ and the other.

Results of the Test Setting

	(1.)								(2.)					(3.)		(4.)	
	member/2	*unpack/2*	*length/2*	*last/2*	*inslast/3*	*switch/2*	*evenpos/2*	*insert/3*	*reverse/2*	*i-sort/2*	*swap/2*	*shift/2*	*multlast/2*	*flatten/2*	*lasts/2*	*odd/1 even/1*	*mergelists/3*
Adate	2.0	1.5	1.2	0.2	2.7	2.8	1.6	16	78	70	232	15	4.3	110	822	—	80
Atre	91.6	×	17.9	6.4	×	1983	156$^{\perp}$	—	—	—	—	—	—	—	—	0.05	—
Dialogs-II	0.03$^{\perp}$	0.05	0.04	0.03$^{\perp}$	0.03	0.19	×	0.06	0.07	0.09$^{\perp}$	0.15	0.11	0.13$^{\perp}$	×	×	—	—

all times in seconds (— not tested × failed ⊥ wrong)

References

[1] A. W. Biermann, G. Guiho, and Y. Kodratoff, editors. *Automatic Program Construction Techniques.* Macmillan, New York, 1984.

[2] M. Berardi D. Malerba, A. Varalro. Learning recursive theories with the separate-and-parallel conquer strategy. In *Proceedings of the Workshop on Advances in Inductive Rule Learning in conjunction with ECML/PKDD*, pages 179–193, 2004.

[3] P. Flener. Inductive logic program synthesis with Dialogs. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 28–51. Stockholm University, Royal Institute of Technology, 1996.

[4] Pierre Flener and Serap Yilmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. *J. Log. Program.*, 41(2-3):141–195, 1999.

[5] E. Kitzelmann and U. Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7(Feb):429–454, 2006.

[6] J. R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 74(1):55–83, 1995.