

Inductive Programming

Example-driven construction of functional programs

Ute Schmid, Martin Hofmann, Emanuel Kitzelmann

We developed an efficient, analytical approach for learning recursive functional programs from examples. Igor2 is a realization of this approach as constructor term rewriting system. We can show that our approach compares very well to other systems of inductive programming and we will discuss applications in the domains of cognitive modelling, end-user programming and automated software engineering.

1 Introduction

Inductive programming research addresses the problem of learning computer programs from incomplete specifications, typically samples of the desired input/output behaviour and possibly additional constraints (Biermann, Guiho, & Kodratoff, 1984; Flener & Schmid, to appear). Induced programs are typically in a declarative – functional or logic – programming language. As a special application of machine learning (Mitchell, 1997), inductive programming creates program *hypotheses*, that is, generalised, typically recursive, programs. In contrast to classification learning, program hypotheses must cover *all* given examples correctly since for programs it is expected that a desired input/output relation holds for all possible inputs. In contrast to deductive approaches to automated program construction, there is no guarantee that the induced program meets the intention of the user. Inductive programming approaches can be characterised by learning biases, especially by a language bias which restricts the syntactic form of programs which can be characterised by a program scheme and by a search bias which determines the sequence in which hypotheses are constructed. In contrast to deductive approaches, the barrier for application is lower since the system user is not required to become an expert in the domain of formal specifications. Therefore, inductive programming approaches are good candidates for the development of programming assistants (Flener & Partridge, 2001).

There are two general approaches to inductive programming: analytical and search-based methods. Search-based methods enumerate syntactically correct programs and test these programs against the given examples guided by some search strategy. The inductive logic programming (ILP) system Foil constructs sets of Horn clauses by sequential covering, that is by a first construct then test approach. This approach was also applied to learning recursive rule sets (Quinlan & Cameron-Jones, 1995). The most successful search-based system is Adate (Olsson, 1995) which constructs ML programs using evolutionary principles. The system MagicHaskell (Katayama, 2005) enumerates Haskell programs with higher-order functions.

Analytical methods are guided by the structure underlying the given input/output examples. Beginning of research on inductive programming in the nineteen-seventies was concerned with such analytical strategies for learning Lisp programs from small sets of positive input/output examples

(Biermann et al., 1984). The most influential of the early systems is Thesys (Summers, 1977). It realized a two step approach to synthesise programs: In a first step, input/output examples were rewritten into traces, in a second step recurrent patterns were searched-for in the traces and the found regularities were generalised to a recursive function. Thesys was restricted to structural problems on lists such as unpacking elements or reverting. Due to that restriction rewriting of input/output examples could be realized with a deterministic algorithm where inputs were characterised by the list structure using *empty* as only predicate. Folding of traces was restricted to linear recursion based on Summers' synthesis theorem which stated that a Kleene sequence of traces can be interpreted as being generated by the *k*th sequence of unfoldings of the searched-for linear recursive program.

For Thesys and all later analytical approaches it is enough to present a small set of only positive input/output examples. These examples must be the first representants of the underlying data-type of the input parameter. In contrast, in search-based approaches, an arbitrary set of positive examples can be presented. Typically, more examples than for analytical approaches are necessary. In addition negative examples can be used to eliminate unsuitable hypotheses.

An extension of Thesys to a larger class of recursive programs is Igor1 (Schmid & Wysotzki, 1998; Schmid, 2003; Kitzelmann & Schmid, 2006). Rewriting of input/output examples allowed nested expressions and folding was guided by a more general program scheme allowing for linear and tree recursions and especially for inducing sets instead of single recursive functions. Another analytical system in the context of ILP is Dialogs (Flener, 1996). Here synthesis relies on interaction with a user who selects a suitable program scheme for the given problem. The most recent analytical approach to inductive programming is Igor2 which will be presented in the following.

2 IGOR2

Igor2 (Kitzelmann & Schmid, 2007; Kitzelmann, 2009, 2008; Kitzelmann & Hofmann, 2008) is a new analytical approach to inductive programming which overcomes inherent limitations of systems in the Thesys tradition, includes powerful concepts introduced in inductive logic programming, and relates to the state of art in functional programming languages. The two step approach of Thesys results in an inherent bot-