ISAR: An Interactive System for Algebraic Implementation Proofs

Bernhard Bauer, Rolf Hennicker
Inst. für Informatik, Ludwig-Maximilians Universität München, Leopoldstr. 11 b, D-8000 München

1 Basic concepts

Formal implementation notions are a necessary prerequisite for proving the correctness of software development steps. In order to be useful in practice formal implementation concepts should be supplied by appropriate proof methods and, even more important, by tools providing mechanical support for correctness proofs. In the following an interactive system, called ISAR, is described which provides an environment for proofs of algebraic implementation relations based on behavioural semantics of equational algebraic specifications. For the basic notions of algebraic specifications, such as signature Σ, term algebra $W_\Sigma(X)$, ground term algebra W_Σ etc. we refer to [2]. Then a *behavioural specification* SP = (Σ, Obs, E) consists of a signature $\Sigma = (S, F)$, a subset Obs $\subseteq$ S of *observable sorts* and a set E of *axioms* (here equations $t = r$ with terms $t, r \in W_\Sigma(X)$).
The definition of our implementation concept is based on the assumption that from the software user's point of view a software product is a correct implementation if it satisfies the desired input/output behaviour. Hence a behavioural specification SP1 = ($\Sigma 1$, Obs1, E1) is called *behavioural implementation* of SP = (Σ, Obs, E) if SP1 respects the observable properties of SP. A precise formal definition of this intuitive notion on the model level is given in [4]. Since we are interested in automatic implementation proofs we will present here only the following proof theoretic characterization for behavioural implementations (cf. [4]) which is the theoretical basis of the ISAR-system. The characterization uses the notion of observable Σ-context which is any term $c[z_s]$ of observable sort $s_0 \in$ Obs over the signature Σ of SP which contains a distinguished variable z_s of some sort $s \in S$. The application of a context $c[z_s]$ to a term t of sort s is defined by the substitution of z_s by t. It is denoted by $c[t]$.

Proof theoretic characterization of behavioural implementations

Let SP1 = ($\Sigma 1$, Obs1, E1) and SP = (Σ, Obs, E) be behavioural specifications such that $\Sigma \subseteq \Sigma 1$ and Obs $\subseteq$ Obs1. SP1 is a behavioural implementation of SP if and only if for all observable Σ-contexts $c[z_s]$ the following property $P(c[z_s])$ is valid:

$P(c[z_s]) = \text{true} \Leftrightarrow$ for all axioms $(t = r) \in E$ and for all ground substitutions $\sigma: X \to W_\Sigma$ the following holds: if t is of sort s then $E1 \vdash \sigma(c[t]) = \sigma(c[r])$.

The usual subterm ordering defines a Noetherian relation on the set of observable Σ-contexts and therefore we can use structural induction for showing the validity of $P(c[z_s])$ for all observable Σ-contexts $c[z_s]$. The automatization of such induction proofs over the structure of contexts (also called *context induction*) is the basic principle of the ISAR-system.

2 Proof of Correct Implementation Steps by the ISAR-System

The input of the ISAR-system is the description of an implementation step consisting of three parts: an abstract specification SP = (Σ, Obs, E) to be implemented, a concrete specification SP-C = (Σ-C, Obs-C, E-C) used as a basis for the implementation and a construction of the implementation. The implementation construction can be defined by some enrichment and/or renaming of SP-C. For instance the following implementation step performs first a renaming of SP-C w.r.t. a signature morphism $\sigma: \Sigma\text{-C} \to \Sigma$ and then an enrichment $\Delta = \langle \Sigma', \text{Obs}', E' \rangle$ of the renamed version of SP-C:

```
implementation step SP_by_SP-C = SP is implemented by SP-C
                                    via renaming σ, enrichment Δ endimplstep
```

Such an implementation step is called *correct* if the normalization of **enrich** (**rename** SP-C **by** σ) **by** $\Delta =_{def} (\sigma(\Sigma\text{-C}) \cup \Sigma', \sigma(\text{Obs-C}) \cup \text{Obs}', \sigma(\text{E-C}) \cup \text{E}')$ is a behavioural implementation of SP. The correctness of implementation steps when performing first an enrichment and then a renaming can be defined analogously.
The normalization of the construction of the implementation is computed by the normalizer module of the ISAR-system (cf. section 3). Since the normalizer can also be used for flattening arbitrary behavioural specifications which are structured by enrichment, renaming and even combination of specifications the abstract specification SP and the concrete specification SP-C can be structured specifications as well. As an example we consider an implementation step which implements a specification SET of finite sets over arbitrary elements on top of a specification LIST of finite lists of elements. For more complicated non-standard examples which, for lack of space, cannot presented here we refer to [1] where e.g. the implementation of stacks by arrays with pointers is proved (using an auxiliary function iterated-pop which defines an iteration of pop operations) and the proof of an implementation of an abstract specification of an imperative programming language by a state-oriented specification of the language is demonstrated. (In the following the variables occurring in the axioms are generated and typed by the system if an implementation step is displayed.)

```
implementation step SET_by_LIST =
    spec SET = enrich ELEM by
      sorts set
      observable sorts bool, elem
      functions emptyset : -> set,
                add : elem, set -> set,
                . is_in . : elem, set -> bool
      axioms   (1) (X32 is_in emptyset) = false,
               (2) (X33 is_in add(X34, X35)) = (eq_elem(X33, X34) or (X33 is_in X35)),
               (3) add(X36, add(X37, X38)) = add(X37, add(X36, X38)),
               (4) add(X39, add(X39, X40)) = add(X39, X40) endspec

  is implemented by

    spec LIST = enrich ELEM by
      sorts list
      observable sorts bool, elem
      functions emptylist : -> list,
                append : elem, list -> list,
                . is_in . : elem, list -> bool
      axioms   (1) (X28 is_in emptylist) = false,
               (2) (X29 is_in append(X30, X31)) = (eq_elem(X29, X30) or (X29 is_in X31)) endspec

  via renaming [set/list], enrichment functions emptyset : -> set,
                                                add : elem, set -> set
                                      axioms    emptyset = emptylist,
                                                add(X41, X42) = append(X41, X42) endimplstep
```

Now the correctness of the implementation step SET_by_LIST can be proved by the ISAR-system using the principle of context induction. The underlying algorithm of the context induction prover of the ISAR-system is described in [5]. For lack of space we do not present here the complete session of the implementation proof but we show the summary of the proof which can be displayed at the end of the session. Thereby any equation occurring as a proof obligation which has been deduced from the axioms of the implementing specification is marked by the comment "* proved". For the proof of equations the ISAR-system is connected to the TIP-system which is an inductive theorem prover (cf. [3]).

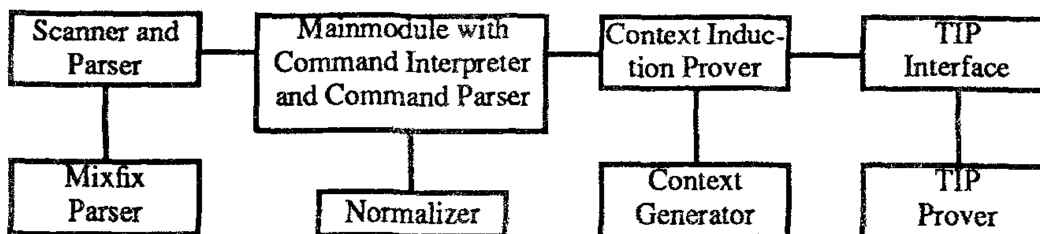
```
****** CONTEXT-INDUCTION ******
Observable Sorts of SET: bool, elem
*** BASE OF THE CONTEXT INDUCTION FOR CONTEXTS OF SORTS: bool, elem
```

* DEPTH: 0
* **PROOF OBLIGATIONS:**
X32 is_in emptyset = false * proved
X33 is_in add(X34, X35) = eq_elem(X33, X34) or X33 is_in X35 * proved
*** **CONTEXT INDUCTION STEP FOR CONTEXTS OF SORTS: bool, elem**
Selected function: . is_in . : elem, set -> bool
* SELECTED CONTEXT: X84 is_in z_set
*** **BASE OF THE CONTEXT INDUCTION FOR CONTEXTS OF SORT: set**
* DEPTH: 1
* **PROOF OBLIGATIONS:**
X84 is_in add(X36, add(X37, X38)) = X84 is_in add(X37, add(X36, X38)) * proved
X84 is_in add(X39, add(X39, X40)) = X84 is_in add(X39, X40) * proved
*** **CONTEXT INDUCTION STEP FOR CONTEXTS OF SORT: set**
Selected function: add : elem, set -> set
* **ADDITIONAL HYPOTHESIS OF THE CONTEXT-INDUCTION:**
X84 is_in constant1_set = X84 is_in constant2_set
* **PROOF OBLIGATIONS:**
X84 is_in add(X363, constant1_set) = X84 is_in add(X363, constant2_set) * proved
* **END OF THE IMPLEMENTATION PROOF: ALL PROOF OBLIGATIONS PROVED** *

3 The Structure of the ISAR-System

The ISAR-system is connected to the TIP-system which verifies all proof obligations generated by the ISAR-system. The main modules of the ISAR-system are

- a scanner and parser with a mixfix-parser for the syntactical analysis of the specifications and implementation steps,
- a normalizer for flattening structured specifications,
- a context induction prover which is the heart of the system for proving the correctness of implementation steps,
- a context generator to produce automatically contexts for context induction steps (the selection of an appropriate context which is general enough for successful termination of implementation proofs has then to be done by the user),
- a TIP-interface for the exchange of informations between the ISAR and the TIP-system
- the proof-modules of the TIP-system for the verification of the proof obligations.



References

1. B. Bauer: Ein interaktives System für algebraische Implementierungsbeweise. Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1992.
2. H. Ehrig, B. Mahr, *Fundamentals of algebraic specification 1*, EATCS Monographs on Theor. Comp. Science **6**, Springer, Berlin, 1985.
3. U. Fraus, H. Hußmann: An inductive theorem prover based on narrowing. Proc. LPAR '92, Logic Programming and Automated Reasoning, Lecture Notes in Artificial Intelligence, Springer, 1992.
4. R. Hennicker: Context induction: a proof principle for behavioural abstractions and algebraic implementations. *Formal Aspects of Computing*, **3** (4), 1991.
5. R. Hennicker: A semi-algorithm for algebraic implementation proofs. Technical Report MIP-9108, Univ. Passau, 1991. Ext. version to appear in: *Theoretical Computer Science*, **104**, 1993.