HOL-Z 2.0:
A Proof Environment for Z-Specifications

Extended Abstract

Achim D. Brucker, Stefan Friedrich, Frank Rittinger, and Burkhart Wolff

Albert-Ludwigs-Universität Freiburg

{brucker,friedric,rittinge,wolff}@informatik.uni-freiburg.de

1 Introduction

The design of tools for formal specification languages (SL) can be roughly divided into two categories: *special purpose design* strives for implementing an SL and its method straight-forwardly in an implementation language. In contrast, *embedded designs* are based on a logical embedding in theorem prover environments such as Isabelle. Examples for the former are Z/EVES, KIV or FDR, examples for the latter are VHDL, HOL-Unity, HOL-CSP and HOL-Z.

The advantage of embedded designs such as HOL-Z (whose underlying conservative embedding into higher-order logic (HOL) has been described in [1]) is its solid logical basis: all symbolic computations on formulas are divided into "logical core theorems" (i.e. derived rules) and special tactical programs controlling their application. Thus, logical consistency of the tool for SL can be reduced to the consistency of the underlying meta-logic and the correctness of the underlying logical engine. The problems with embedded designs are threefold:

1. A logical embedding must be designed for application — this usually conflicts with other design goals such as proof of meta-theoretic properties.
2. Embeddings often present the embedded language in the form of meta-logical formulas: this has effects on syntax, error-handling, and proof style.
3. The concrete prover sometimes enforces a particular organization of specifications which is unsuited for larger developments (e.g. bottom-up).

In order to meet these problems, we improved our logical embedding for Z in Isabelle/HOL called HOL-Z. Our main contribution in this paper consists of an integration of HOL-Z into a specific tool-chain. The integrated environment — still called HOL-Z for simplicity — offers the following features:

1. HOL-Z is based on a "shallow embedding" [1]; many elements of Z are "parsed away" and represent no obstacle for symbolic manipulations,
2. HOL-Z is based on a new front-end consisting of a common editor with an integrated parser and type checker; this paves the way for high-level error-messages and for professional documentation,
3. HOL-Z offers technical support of methodology (such as refinement, top-down proof development, proof-obligation management), and support of particular "proof-idioms" such as the schema calculus in Z.

2 A Tool Chain for Literate Specification

HOL-Z is now embedded in a chain of tools, that can either be integrated into XEmacs (which is our preferred setting since, for example, a click on a type error messages leads to a highlighting of the corresponding source) or in usual shell scripts, that allow for an easy integration of the specification process into the general software development process. The data flow in our tool chain can be

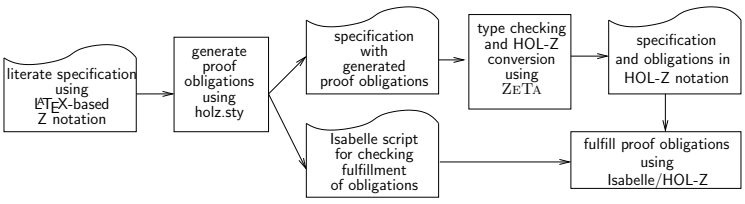

Fig. 1. A Tool Chain supporting Literate Specification

described as follows: At the beginning, a normal LaTeX-based Z specification is created. Running LaTeX leads to the expansion of proof-obligation macros, which also generates an Isabelle-script that checks that the obligations are fulfilled (to be run at a later stage). ZeTa takes over, extracts the definitions and axioms from the LaTeX source (including the generated ones) and type checks them or provides animation for some Z schemas. Our plug-in into ZeTa converts the specification (sections, declarations, definitions, schemas, ...) into SML-files that can be loaded into Isabelle. In the theory contexts provided by these files, usual Isabelle proof-scripts can be developed. An integration of the process into a version management system allows for semantically checked specifications: for example, only when the proof obligation check scripts run successfully, new versions of the specification document are accepted as final versions.

2.1 `holz.sty` — A Macro Package for Generating Proof Obligations

We decided to use LaTeX itself as a flexible mechanism to construct and present proof-obligations inside the specification — this may include consistency conditions, refinement conditions or special safety properties imposed by a special formal method for a certain specification architecture. Our LaTeX-package `holz.sty` provides, among others, commands for generating refinement conditions as described in [2], where also the paradigmatical "BirthdayBook" is presented we use as running example. For *AddBirthday is refined by AddBirthday*1, we instantiate a macro as follows:

```
\zrefinesOp[Astate=BirthdayBook, Cstate=BirthdayBook1,
            Aop=AddBirthday, Cop=AddBirthday1,
            Args={name?: NAME; date?: DATE}, Abs=Abs]{Add}
```

Here, `Astate` contains the schema describing the abstract state and `Cstate` hold the schema describing the concrete state. Based on this input, our LaTeX-package automatically generates the following two proof obligations:

$$\begin{aligned} Add_1 == & \forall\, BirthdayBook;\ BirthdayBook1;\ name? : NAME;\ date? : DATE \bullet \\ & (\mathsf{pre}\ AddBirthday \land Abs) \Rightarrow \mathsf{pre}\ AddBirthday1 \\ Add_2 == & \forall\, BirthdayBook;\ BirthdayBook1;\ BirthdayBook1';\ name? : NAME; \\ & date? : DATE \bullet (\mathsf{pre}\ AddBirthday \land Abs \land AddBirthday1) \\ & \Rightarrow (\exists\, BirthdayBook' \bullet Abs' \land AddBirthday) \end{aligned}$$

These proof obligations are type checked using ZETA and are converted to HOL-Z by our ZETA-to-HOL-Z converter.

2.2 The ZETA System

ZETA [3] is an open environment for the development, analysis and animation of specifications based on Z. Specification documents are represented by *units* in the ZETA system, that can be annotated with different *content* like LaTeX mark-up, type-checked abstract syntax, etc. The contents of units is computed by adaptors, which can be plugged into the system dynamically.

2.3 ZETA-to-HOL-Z Converter

The converter consists of two parts: an adaptor that is plugged into ZETA and converts the type-checked abstract syntax of a unit more or less directly into an SML file. On the SML side, this file is read and a theory context is build inside Isabelle/HOL-Z. This involves an own type-checking and an own check of integrity conditions of the specification and some optimizations for partial function application in order to simplify later theorem proving.

In its present state, the converter can translate most Z constructs with the exception of user-defined generic definitions, arbitrary free types or less frequently used schema operators like hiding and piping.

2.4 Experiences with Case-Studies

We applied HOL-Z to several specifications, including Spivey's Birthday Book, an architecture of CVS (the Concurrent Versions System) and the CORBA Security Service, with a focus on security analysis of CVS and CORBA.

We applied our tool-chain including some proofs of refinement conditions for the Birthday Book example and some proofs of the refinement of an abstract architectural description of CVS to the implementation on top of a Unix file system. The large CORBA example (approx. 90 pages (!), that are converted and loaded in less than 5 minutes on a standard PC under PolyML) shows the feasibility of our approach for real world examples.

3 Proof Support for Z

3.1 Isabelle/HOL-Z revisited

The SL Z is centered around a specific structuring mechanism called *schema*. Semantically, schemas are just sets of records (called *bindings* in Z terminology; [4]) of a certain type. Z is based on typed set theory equivalent to HOL set theory. However, a reference to a schema can play different *roles* in a specification: it can serve as *import* in the declaration list in other schemas, or as *predicate* (where all arguments are suppressed syntactically), or as *set* (see [1] for more details).

The approach of HOL-Z is to represent records by products in Isabelle/HOL and to manage their layout in order to support *as import*-references. This is achieved by a parser making implicit bindings in Z expressions explicit and generating coercions of schemas according to their role. For example, we assume throughout this section a schema A of type $[x_1 \mapsto \tau_1, x_2 \mapsto \tau_2]$ and a schema B of type $[x_2 \mapsto \tau_2, x_3 \mapsto \tau_3]$. Then, a schema expression $A \wedge B$ can be represented by

$$\lambda(x_1, x_2, x_3) \bullet A(x_1, x_2) \wedge B(x_2, x_3) \ ,$$

while an expression $A \cup A$ will be represented by $(\text{asSet } A) \cup (\text{asSet } A)$.

Thus, having "parsed away" the specific binding conventions of Z into standard λ-calculus, Isabelle's proof-engine can handle Z as ordinary HOL-formulas. There is no more "embedding specific" overhead such as predicates stating the well-typedness of certain expressions, etc.

For full-automatic proofs this is fine; however, in practice, realistic case studies require proofs with user interaction. This leads to the requirement that intermediate lemmas can be inserted "in the way of Z", intermediate results are presented "Z-alike" and deduction attempts to mimic the proof style imposed by Z (cf. [5]). As a prerequisite, we defined a special abstraction operator *SB* semantically equivalent to the pair-splitting λ-abstraction from the example above, which is actually encoded by:

$$\text{SB } \text{“}x_1\text{”} \rightsquigarrow x_1, \text{“}x_2\text{”} \rightsquigarrow x_2, \text{“}x_3\text{”} \rightsquigarrow x_3 \bullet A(x_1, x_2) \wedge B(x_2, x_3)$$

where each field-name is kept as a (semantically irrelevant) string in the representation. Thus, while the "real binding" is dealt with by Isabelle's internal λ, which is underlying α-conversion, the *presentation* of intermediate results is done on the basis of the original field-names used in the users specification.

3.2 New Proof Support in HOL-Z 2.0

Schemas can also be used in quantifications as part of some very Z specific concept, the so-called *schema calculus*, for which we implemented syntax and proof support. For example, $\forall A \bullet B$ is a schema of type $\mathbb{P}([x_3 \mapsto \tau_3])$. In HOL-Z, it is represented by:

$$\text{SB } \text{”}x_3\text{”} \rightsquigarrow x_3 \bullet \forall(x_1, x_2) : \text{asSet } A \bullet B(x_2, x_3)$$

This and similar quantifiers and operators allow for a very compact presentation of typical proof-obligations occurring in refinements in Z. As example, we use an already slightly simplified version of Add_1 already described in [2, pp. 138]. (The full proof had been omitted for space reasons). Instead of referring to constants representing the proof obligations generated by the LaTeX-based front-end, we use the HOL-Z-parser directly:

```
zgoal thy
"∀ BirthdayBook •∀  BirthdayBook1 •∀  name? ∈ Name •∀ date? ∈ Date •
     (name? ∉ known ∧ known = {n. ∃ i: #1..hwm. n=names i}
      =+=> (∀ i  : #1..hwm. name? ≠names i))";
```

which opens an Isabelle proof-state.

In the literature, several calculi for the schema calculus have been presented more or less formally ([4, 6]). From the perspective of HOL-Z it is quite clear what is needed: for any construct of the schema calculus, a special tactic must be provided that works analogously to the usual introduction and elimination rules for standard (bounded) quantifiers and set comprehensions. These tactics have been implemented and combined to new tactics, for example to a tactic that "strips-off" all universal quantifiers (including schema quantifiers) and implications. Thus, the HOL-Z tactic:

```
by(stripS_tac 1);
```

transforms the goal into the following proof state:

```
1. !!birthday known dates hwm names name? date? i.
    [| BirthdayBook (birthday, known); BirthdayBook1 (dates, hwm, names);
       name? : Name; date? : Date;
       name? ∉ known ∧ known = {n. ∃ i: #1..hwm. n=names i};
       i : ( #1 .. hwm) |] ⇒name? ≠names i
```

Note that the quite substantial reconstruction of the underlying binding still leads to a proof state that is similar in style and presentation to [5].

Besides the "schema calculus", Z comes with a large library of set operators specifying relations, functions as relations, sequences and bags; this library — called the *Mathematical Toolkit* of Z — differs in style substantially from the Isabelle/HOL library, albeit based on the same foundations. For HOL-Z 2.0, we improved this library substantially and added many derived rules that make a higher degree of automatic reasoning by Isabelle's standard proof procedures possible. For example, the goal above is simply "blown away" by:

```
auto();
```

which finishes the proof.

4 Conclusion and Further Work

We have presented HOL-Z, a tool chain for writing Z specifications, type-checking them, and proving properties about them. In this new setting, we can specify our Z specifications in a type setting system, automatically generate proof obligations, import both of them into a theorem prover environment and use the existing proof mechanisms to gain a higher degree of automation. With the proof support for the schema calculus, realistic analysis of specifications, in particular refinement proofs become feasible.

A consequence of our implementation of the converter is that we cannot directly interact between ZeTa and HOL-Z. A closer integration of HOL-Z into ZeTa would be desirable but has not been realized so far.

We will investigate if the introduction and elimination tactics can be integrated much deeper into Isabelle's `fast_tac` procedure; this would pave the way for a tableaux-based approach of reasoning over the "schema calculus" — which is, to our knowledge, a new technique for automated deduction on Z specifications.

References

1. Kolyang, Santen, T., Wolff, B.: A structure preserving encoding of z in isabelle/hol. In von Wright, J., Grundy, J., Harrison, J., eds.: Theorem Proving in Higher Order Logics — 9th International Conference. LNCS 1125, Springer Verlag (1996) 283–298
2. Spivey, J.M.: The Z Notation: A Reference Manual. 2nd edn. Prentice Hall International Series in Computer Science (1992)
3. anonymous: The ZeTa system (2002) `http://uebb.cs.tu-berlin.de/zeta/`.
4. anonymous: Formal Specification – Z Notation – Syntax, Type and Semantics (2000) Consensus Working Draft 2.6.
5. Woodock, J., Davies, J.: Using Z. Prentice Hall (1996)
6. Henson, M.C., Reeves, S.: A logic for the schema calculus. In Bowen, J.P., Fett, A., Hinchey, M.G., eds.: ZUM'98: The Z Formal Specification Notation. Volume 1493 of Lecture Notes in Computer Science., Springer-Verlag (1998) 172–191