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

Achim D. Brucker
Albert-Ludwigs-Universität Freiburg
`brucker@informatik.uni-freiburg.de`

Frank Rittinger
Albert-Ludwigs-Universität Freiburg
`rittinge@informatik.uni-freiburg.de`

Burkhart Wolff
Albert-Ludwigs-Universität Freiburg
`wolff@informatik.uni-freiburg.de`

Abstract: We present a new proof environment for the specification language Z. The basis is a semantic representation of Z in a structure-preserving, shallow embedding in Isabelle/HOL. On top of the embedding, new proof support for the Z schema calculus and for proof structuring are developed. Thus, we integrate Z into a well-known and trusted theorem prover with advanced deduction technology such as higher-order rewriting, tableaux-based provers and arithmetic decision procedures. A further achievement of this work is the integration of our embedding into a new tool-chain providing a Z-oriented type checker, documentation facilities and macro support for refinement proofs; as a result, the gap has been closed between a logical embedding proven correct and a *tool* suited for applications of non-trivial size.
Key Words: Theorem Proving, Refinement, Z

Category: D.2.1, D.2.4, F.3.1, F.4.1

1 Introduction

Tools for formal specification languages can roughly be divided into two categories: *straightforward design* which implements a specification environment directly in a programming language, and *embedded design* which implements it on the basis of a logical embedding in a theorem prover environment, e.g. Isabelle [Paulson, 1994]. Examples of the former are Z/EVES [ZEVES, 2003], KIV [KIV, 2003] or FDR [FDR, 2003], examples of the latter are VHDL [Reetz, 1995], HOL-Unity [Paulson, 2000], HOL-CSP [Tej and Wolff, 1997] and HOL-OCL [Brucker and Wolff, 2002].

The advantage of embedded designs such as HOL-Z (whose underlying conservative embedding of Z into the higher-order logic (HOL) instance of Isabelle has been described in [Kolyang et al., 1996]) is their solid logical basis: all symbolic computations on formulae are divided into "logical core theorems" (i.e. derived rules) and special tactical programs controlling their application. Thus, logical consistency of a tool for specification languages can be reduced to the consistency of the underlying meta-logic and the correctness of the underlying logical engine, which is in our case a well-known and accepted one. When scaling up to a tool, the problems with embedded designs are threefold:

1. A tool-oriented logical embedding must be designed for *effective deduction*. This usually conflicts with other design goals such as provability of meta-theoretic properties (e.g. completeness).

2. Embeddings often present the embedded language in the form of meta-logical formulae: this has negative effects on presentation and error-handling.

3. The embedding and the concrete prover may suggest unstructured proof attempts ("unfold everything into meta-logic, bust the pieces there . . . ") and an unnatural proof organization. This may be too low-level for larger developments.

In order to meet these problems, we improved our logical embedding called HOL-Z. The integrated environment — still called HOL-Z for simplicity — offers the following features:

1. HOL-Z is a "shallow embedding" [Kolyang et al., 1996]; types are handled on the meta-level, and many elements of Z are "parsed away" and represent no obstacle for deduction.

2. HOL-Z is based on a new front-end consisting of an integrated parser and type checker; this paves the way for professional documentation and high-level error-handling.

3. HOL-Z offers technical support of methodology (such as refinement, top-down proof development or proof obligation management), and support of particular "structured proof idioms" such as the schema calculus in Z.

Our first contribution in this paper consists in a proof calculus for the schema calculus of Z and its implementation based on derived rules. As the second contribution, we provide an integration of HOL-Z into a specific tool-chain in order to "scale up" the previous work on embedding Z into Isabelle/HOL to a proof environment that has been applied in several larger case studies.

2 Foundations

2.1 Isabelle/HOL

Higher-order logic (HOL) [Church, 1940; Andrews, 1986] is a classical logic with equality enriched by total polymorphic higher-order functions. It is more expressive than first-order logic, since e.g. induction schemes can be expressed inside the logic. Pragmatically, HOL can be viewed as a combination of a typed functional programming language like Standard ML (SML) or Haskell extended by logical quantifiers.

When extending logics, two approaches can be distinguished: the *axiomatic method* on the one hand and *conservative extensions* on the other. Extending HOL via axioms easily leads to inconsistency; given the fact that libraries contain several thousand theorems and lemmas, the axiomatic approach is too error-prone in practice. In contrast, a conservative extension introduces new constants (by *constant definitions*) and types (by *type definitions*) only via axioms of a particular form; a proof that conservative extensions preserve consistency can be found in [Gordon and Melham, 1993].

The HOL library provides conservative theories for the HOL-core based on type *bool*, for the numbers such as *nat* and *int*, for typed set theory based on τ *set* and a list theory based on τ *list*.

Isabelle [Paulson, 1994] is a *generic* theorem prover. New object logics can be introduced by specifying their syntax and inference rules. Among other logics, Isabelle supports first-order logic (intuitionistic and classical), Zermelo-Fränkel set theory (ZF) and HOL, which we choose as a framework for HOL-Z.

Following the tradition of LCF-style provers, Isabelle consists of a logical engine encapsulated in an abstract data type *thm* in SML; any *thm* object has been constructed by trusted elementary rules in the kernel. Thus Isabelle supports user-programmable extensions in a logically safe way. A number of generic proof procedures (*tactics*), written in SML, have been developed. A special tactic is the simplifier based on higher-order rewriting and proof-search procedures based on higher-order resolution.

2.2 Z by Example

The formal specification language Z [Spivey, 1992] is based on typed set theory and first-order logic with equality. The syntax and the semantics are specified in an ISO-standard [ISOZ, 2002]; for future standardization efforts of operating system libraries or programming language semantics, Z is therefore a likely candidate. Z provides constructs for structuring and combining data-oriented specifications: schemas model the states of the system (*state schemas*) and operations on states (*operation schemas*), while the *schema calculus* is used to compose these sub-specifications to larger ones. We present these constructs using a standard example, Spivey's "birthday book". This simple system stores names and dates of birthdays and provides, for example, an operation to add a new birthday. In Z, abstract types for $NAME$ and $DATE$ can be declared that we use in a schema (consisting of a declaration part and a predicate part) to define the system state $BirthdayBook$. For transitions over the system state, the schema $AddBirthday$ is used:

$$\begin{array}{|l}
\multicolumn{1}{l}{BirthdayBook}\\ \hline
known : \mathbb{P}\, NAME\\
birthday : NAME \rightarrowtail\!\!\!\!\!\!+\;\; DATE\\ \hline
known = \mathsf{dom}\; birthday\\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{AddBirthday}\\ \hline
\Delta BirthdayBook\\
n? : NAME;\;\; d? : DATE\\ \hline
n? \notin known\\
birthday' = birthday \cup \{n? \mapsto d?\}\\ \hline
\end{array}$$

$\Delta BirthdayBook$ imports the state schema into the operation schema in a "stroked" and a "non-stroked" version: $BirthdayBook'$ and $BirthdayBook$. The resulting variables $birthday'$ and $birthday$ are conventionally understood as the states after and before the operation, respectively.

This system is refined to a more concrete one based on a state $BirthdayBook1$ containing two (unbounded) arrays and an operation that implements $AddBirthday$ on this state:

$$\begin{array}{|l}
\multicolumn{1}{l}{BirthdayBook1}\\ \hline
names : \mathbb{N} \rightarrowtail\!\!\!\!\!\!+\;\; NAME\\
dates : \mathbb{N} \rightarrowtail\!\!\!\!\!\!+\;\; DATE\\
hwm : \mathbb{N}\\ \hline
\forall\, i, j : 1 \,..\, hwm \bullet i \neq j\\
\qquad \Rightarrow names(i) \neq names(j)\\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{AddBirthday1}\\ \hline
\Delta BirthdayBook1\\
name? : NAME;\;\; date? : DATE\\ \hline
\forall\, i : 1 \,..\, hwm \bullet name? \neq names(i)\\
hwm' = hwm + 1\\
names' = names \oplus \{hwm' \mapsto name?\}\\
dates' = dates \oplus \{hwm' \mapsto date?\}\\ \hline
\end{array}$$

The relation between abstract states (captured by the schema $Birthday$) and the concrete states (captured by $Birthday1$) is again represented in a schema in Z, namely

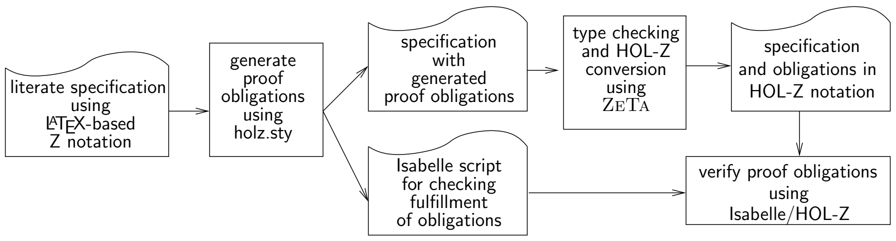
literate specification using LaTeX-based Z notation
generate proof obligations using holz.sty
specification with generated proof obligations
Isabelle script for checking fulfillment of obligations
type checking and HOL-Z conversion using ZeTa
specification and obligations in HOL-Z notation
verify proof obligations using Isabelle/HOL-Z

Figure 1: A Tool Chain supporting Literate Specification

the schema *Abs*; the relation defines *known* as the range of the *names*-array (upto high-water-mark *hwm*) and the relation from positionally associated *names* and *dates* as equal to the relation *birthday* from the abstract state *Birthday*:

```
── Abs ──────────────────────────────
│ BirthdayBook
│ BirthdayBook1
├──────────────────
│ known = {i : 1 .. hwm • names(i)}
│ ∀ i : 1 .. hwm • birthday(names(i)) = dates(i)
└─────────────────────────────────────
```

One can use the schema calculus to combine different operation schemas into one operation. For example, one could strengthen the *AddBirthday* operation with an operation schema *AlreadyKnown* which expresses the fact that the entry that should be added already exists in the birthday book:

$$Add == AddBirthday \lor AlreadyKnown$$

The birthday book will be our running example throughout the rest of the paper.

3 A Tool Chain for Literate Specification

The core of HOL-Z, namely the logical embedding discussed in the next chapters, is now integrated into a chain of tools. We briefly describe the data flow of our tool-chain as depicted in Fig. 1; in the following sections we will describe the components of the tool-chain in more detail.

At the beginning, a normal LaTeX-based Z specification is created; the specification may contain formal text, macros for proof obligation generation and informal explanations in a mixed, “literate specification” style. Running LaTeX leads to the expansion of proof obligation macros, and also generates an Isabelle-script that checks that the obligations are fulfilled (to be run at a later stage). ZeTa takes over, extracts all Z definitions 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.

The elements of our tool chain can be technically organized in various ways. One way is to build a front-end by integrating ZeTa into XEmacs (which is our preferred setting since, for example, a click on a type-error message leads to a highlighting of the corresponding source) and a back-end based on Isabelle. Another way is an organization into usual shell scripts, that allows for easy integration of the specification process into the general software development process, including in particular version management that allows for semantically checked specifications. In this setting, for example, one can assure that new versions of the specification document are accepted as main versions only when the proof obligation check scripts run successfully, etc.

3.1 The LaTeX-based Z Specification

The formal text in a specification document closely follows the LaTeX format of the Z standard described in [ISOZ, 2002]. In this section, we therefore focus on our add-on `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 method for a certain specification architecture. Our LaTeX package `holz.sty` provides, among others, commands for generating refinement conditions as described in [Spivey, 1992]. For our running example of the birthday book's *AddBirthday* operation, we instantiate the *refinement condition* that *AddBirthday* is refined by the more concrete *AddBirthday*1 as follows:

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

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

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

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

3.2 The ZeTa System

ZeTa [Zeta, 2003] is an open environment for the development, analysis and animation of specifications. 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 system is aware of dependencies between the units and attempts to exploit this when units change. An integration of ZeTa into the editing environment XEmacs greatly facilitates changes and the management of consistency checking in large specifications. The contents of units is computed by adaptors, which can be plugged into the system dynamically.

Two plugins are available that are particularly important for our purpose: one consists in a *type checker* for Z based on LaTeX covering a large part of the Z standard; the other is an *animator* for Z that allows for the evaluation of Z expressions, in particular schemas. Thus, specifications can be tested easily during the specification work helping to avoid spurious errors.

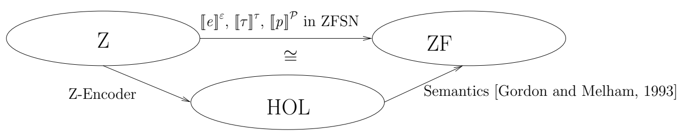
$[\![e]\!]^{\varepsilon}$, $[\![\tau]\!]^{\tau}$, $[\![p]\!]^{\mathcal{P}}$ in ZFSN
Z
ZF
≅
Z-Encoder
HOL
Semantics [Gordon and Melham, 1993]

Figure 2: An Overview of Semantic Relations

3.3 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 built inside Isabelle/HOL-Z. This involves the conversion into the internal HOL-Z representation by the *Z-Encoder* (see Sec. 4.1), followed by an independent type checking of the result by Isabelle (ruling out that implementation errors in the Z-Encoder may yield inconsistency), followed by a check of conservativity conditions for schemas 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.

4 Representing Z in Isabelle/HOL: The Foundations

In order to be self-contained, we present the foundations of the HOL-Z embedding. While most basic concepts of this embedding have been developed by one of the authors jointly with Santen and Kolyang [Kolyang et al., 1996], the implementation of the new "Z-Encoder" is a complete redevelopment; this also involves new machinery for converting types, bindings, and schema calculus constructs.

4.1 Conformance with "The Standard"

For any embedding of a logic, the question of the *faithfulness of the encoding of one calculus in another* has to be raised. This question seems to be very critical for HOL-Z since the semantics in the Z standard [ISOZ, 2002] (ZFSN) is based on Zermelo-Fränkel set theory (ZF) and not on typed set theory as HOL. ZFSN does not define a deductive system: It provides a semantics in set theory and requires "conformance" of a *deductive system* for Z, i.e. the soundness of all rules of the system with this semantics.

The core of the ZFSN semantics consists of the definition of the *partial* functions $[\![\tau]\!]^{\tau}$, $[\![e]\!]^{\varepsilon}$ and $[\![p]\!]^{\mathcal{P}}$ that assign to each element of each syntactic category (types τ, expressions e and predicates p) a *type* resp. a *value* (meaning). A calculus *conforms to the standard* if it reflects the semantic function *where it is defined*. The semantic functions are interpreted in an untyped universe of ZF. In the semantic universe, objects like $\{0, \{0\}\}$ may occur that are illegal in the typed set theory of HOL. This does not mean that $\{0, \{0\}\}$ is legal in Z; in fact, one of the major objectives of $[\![\tau]\!]^{\tau}$ is to rule out such expressions by a type-discipline that can be injectively mapped into the typed λ-calculus underlying HOL.

Fig. 2 outlines the semantic situation: Let HOL_{τ} be the set of HOL-terms of simple type τ_{λ}. Moreover, let Z_{τ} denote the set of Z-expressions of a Z-type τ_Z, and *ZF* the class of sets in ZF into which all elements of Z_τ are mapped. The two type systems are both interpreted in a universe, i.e. a ZF-set. According to ZFSN, the Z universe is closed under Cartesian products and powerset-construction. According to [Gordon and Melham, 1993], the HOL universe is a set closed under function construction $A \to B$ and the type *bool*. The crucial point for the correctness of the overall approach is that both universes are *isomorphic*. Slightly simplified, the types τ_Z and τ_λ are defined as:

$$\begin{aligned} \tau_Z = {} & \text{integer} \\ & \mid \tau_Z \times \cdots \times \tau_Z \\ & \mid \langle\!| \; \tau_{n_1} \rightsquigarrow \tau_z, \ldots, \tau_{n_n} \rightsquigarrow \tau_Z \; |\!\rangle \\ & \mid \mathbb{P}\, \tau_Z \end{aligned} \qquad \begin{aligned} \tau_\lambda = {} & \text{bool} \\ & \mid \text{integer} \\ & \mid \tau_\lambda \times \tau_\lambda \\ & \mid \tau_\lambda \to \tau_\lambda \end{aligned}$$

The injection from τ_Z to τ_λ is now defined as follows: integers are mapped to themselves, multiple Cartesian products $\tau_Z \times \cdots \times \tau_Z$ to binary products associated to the right, bindings (i.e. records) $\langle\!| \; \tau_{n_1} \rightsquigarrow \tau_Z, \ldots, \tau_{n_n} \rightsquigarrow \tau_Z \; |\!\rangle$ to n-ary Cartesian products sorted by their *tag names* τ_{n_i}, and $\mathbb{P}\, \tau_Z$ to $\tau_\lambda \to$ *bool*. From this mapping, it can be seen that HOL-Z is slightly more liberal than τ_Z (it admits mixtures of Cartesian products and bindings that are not allowed in Z, for example), but the semantic domains are still isomorphic to each other. In particular, the typed set theory of Z is converted to the theory of typed characteristic functions in HOL[1]. Thus, the *Z-Encoder* maps all terms in Z_τ to specific HOL_τ-terms, such that the diagram in Fig. 2 commutes up to isomorphism for all τ. Our argument works for a monomorphic type universe. For more details and an extension to polymorphic types, see [Santen, 1998].

It is perhaps surprising to discover that the semantic basis of Z as described in the rather complex ZFSN is just an equivalent to the standard model of the typed λ-calculus. It remains to evaluate the syntactical, *notational* facet of Z can be handled by our *Z-Encoder* (to be discussed in the next section).

4.2 Encoding Schemas

Semantically, schemas are just sets of *bindings* of a certain type. However, a reference to a schema can play different *roles* in a specification: it can serve as *import* in the declaration list of other schemas (e.g. reference A in schema B in Fig. 3), it can serve as *set* (e.g. reference A or B in schema C in Fig. 3), or it can serve as *predicate* in the so-called schema calculus (see below). Moreover, references to schemas may be *decorated* by a stroke, which results in a renaming of the variables in a schema and of the tag names in the corresponding schema type by suffixing them with a stroke. Schema operators like ΔA are syntactic synonyms for $A \wedge A'$. Note that several occurrences of declarations (e.g. x_2 in schema B) in a schema are *identified* and their associated sets S_2 and T are intersected (provided that their underlying types are equal; otherwise, the whole declaration is illegal). It is this particular feature of Z that excludes a treatment of schemas by sets of "extensible records" [Naraschewski and Wenzel, 1998; Brucker and Wolff, 2002].

The first key idea for the design of HOL-Z is to compute a raw type, the *schema-signature* S_Σ for all expressions of schema type. More precisely, a schema-signature $S_\Sigma\, \mathbb{P} \langle\!| \; x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n \; |\!\rangle$ is just the ordered list of tag names $[x_1, \ldots, x_n]$. In the Z-Encoder, schema-signatures are abstracted from τ_Z-types which are available whenever

[1] In our implementation, however, the situation is slightly more complex: the above said is true for schema types $\mathbb{P}\langle\!| \; \tau_{n_1} \rightsquigarrow \tau_Z, \ldots, \tau_{n_n} \rightsquigarrow \tau_Z \; |\!\rangle$; all other $\mathbb{P}$-types are mapped to the type-constructor *set* — which is defined isomorphic to characteristic functions, but distinguished from them by the type-system of HOL. This optimization gives better access to HOL-libraries and a bit more type-safety in HOL-Z.

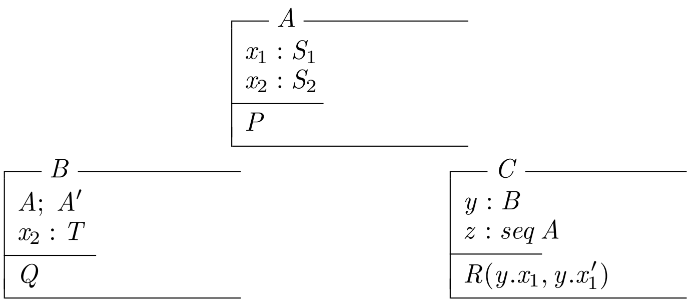

Figure 3: Schemas and their Use

ZeTa is used as front-end. When using the weaker typed e-mail format (as in locally stated proof-goals; see Sec. 5.5), they are approximated on the basis on schema syntax, on previously compiled schemas from an environment, and on computations of the effect of schema operators over schema subexpressions.

The second key idea is to represent schemas "as predicates" by default, i.e. as characteristic functions over bindings, that are represented as products. This is achieved by a pre-translator on parsed terms in the Z-Encoder, that makes implicit bindings in Z expressions — expressed by their schema-signature — explicit and generates coercions of schemas according to their role. For example, the schema declaration A, as depicted in Fig. 3, of type $\mathbb{P}\langle\!\langle\, x_1 \mapsto \tau_1, x_2 \mapsto \tau_2 \,\rangle\!\rangle$, is converted into the constant definition:

$$A \equiv \lambda(x_1, x_2) \bullet x_1 \in S_1 \land x_2 \in S_2 \land P$$

As a result of this presentation, the treatment of schema references as import or as set, as used in schema B, can be represented truthfully as follows:

$$B \equiv \lambda(x_1, x_2, x_1', x_2') \bullet A(x_1, x_2) \land A(x_1', x_2') \land x_2 \in T \land Q$$

while an expression $A \cup A$ will be represented by (asSet A)$\cup$(asSet A) (with the coercion asSet from characteristic functions to typed sets in HOL).

As mentioned, there is a further role in which schema references may be used: in the schema calculus, one may write a schema expression $A \land B$ which has the schema type $\mathbb{P}\langle\!\langle\, x_1 \mapsto \tau_1, x_2 \mapsto \tau_2, x_1' \mapsto \tau_1, x_2' \mapsto \tau_2 \,\rangle\!\rangle$. Such an expression will be represented by

$$\lambda(x_1, x_2, x_1', x_2') \bullet A(x_1, x_2) \land B(x_1, x_2, x_1', x_2')$$

The schema type of the conjunction of two schema expressions is the *union* of the schema signatures, provided that each tag name is associated with the same type. Thus, having "parsed away" the specific binding conventions of Z into standard λ-calculus, Isabelle's proof-engine can handle Z as ordinary HOL-formulae. There is no further "embedding specific" overhead such as predicates stating the well-typedness of certain expressions, etc; these issues are handled inside the typing discipline of HOL. Note, moreover, that our representation keeps the structure of the original Z specification — previous attempts [Bowen and Gordon, 1995] had been based on "flattening" (unfolding) of the schema notation — and allows for a controlled unfolding of schemas in the course of a proof.

So far, the presentation of binding is adequate for automatic proofs; 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 the proof style imposed by Z (cf. [Woodock and Davies, 1996]) can be mimicked. Therefore, we define a special “annotated” abstraction operator *SBinder* semantically equivalent to the pair-splitting λ-abstraction from the example above:

```
consts   SBinder0          ::   "[string , β ⇒ δ] ⇒ (β ⇒ δ)"
         "SBinder0 An P  ≡   P"
         SBinder           ::   "[string ,[β,γ] ⇒ δ] ⇒ ((β ∗ γ)⇒δ)"
         "SBinder An P   ≡   (λ (x,y). (P x y))"
```

and introduce the notation:

$$\text{SB “}x\text{”} \leadsto x,\ \text{“}y\text{”} \leadsto y,\ \text{“}z\text{”} \leadsto z.P$$

for:

$$\text{SBinder “}x\text{”}(\lambda\, x.\text{SBinder “}y\text{”}(\lambda\, y.\text{SBinder0“}z\text{”}(\lambda\, z.P)))$$

Using these operators, the example above is pretty-printed by:

$$\text{SB “}x_1\text{”} \leadsto x_1,\ \text{“}x_2\text{”} \leadsto x_2,\ \text{“}x_1'\text{”} \leadsto x_3,\ \text{“}x_2'\text{”} \leadsto x_4 \bullet A(x_1,x_2) \wedge B(x_1,x_2,x_3,x_4)$$

where each field name is kept as a (semantically irrelevant) string in the representation. Thus, while the “real binding” is dealt by Isabelle’s internal λ, which is subject to α-conversion, the *presentation* of intermediate results is done on the basis of the original field-names from the user’s specification.

5 Proof Support for Z

Based on the semantic representation of Z in Isabelle/HOL presented in the previous section, we will now describe structured proof-support for Z.

5.1 The Mathematical Toolkit of Z

Z comes with a large toolkit of mathematical definitions concerning relations, functions, sets and bags one can build on when specifying software systems. Based on the observation that the semantic domains are equivalent, it is now straightforward to embed this “Mathematical Toolkit” conservatively in HOL.

The type of *relation* (written A↔B) is defined as the set of all pairs over A and B. Thus, in contrast to HOL, all functions are encoded by their graph. This allows for partial function spaces and for operations like the union of two functions.

The toolkit is presented as a suite of constant definitions (the technique is equivalent to [Bowen and Gordon, 1995]). On the right-hand side of the type definition some parsing information is given together with the binding values.

```
consts
    partial_func   ::"[α set,β set] ⇒ (α ↔ β) set"   ("_  ⇸  _"      [54,53] 53)
    total_func     ::"[α set,β set] ⇒ (α ↔ β) set"   ("_  →  _"      [54,53] 53)
    partial_inj    ::"[α set,β set] ⇒ (α ↔ β) set"   ("_  ⤔  _"      [54,53] 53)
    func_overrid   ::"[α ↔ β, α ↔ β] ⇒ (α ↔ β)"      ("_ '(+') _"    [55,56] 55)
defs
    total_func_def     "S → R   ≡{s. s ∈ S ⇸ R ∧ dom s= S}"
    partial_inj_def    "S ⤔ R   ≡{s. s ∈ S ⇸ R ∧ (∀ x₁x₂y. (x₁,y) ∈s
                                       ∧ (x₂,y) ∈s → x₁= x₂)}"
    func_overrid_def   "S (+) R ≡(dom R ⩤ S) ∪ R"
```

Conformance of the set operators $\nrightarrow, \rightarrow, \ldots, \oplus$ of the mathematical toolkit is easy to verify: Just compare these definitions (and there are hundreds) with the ones in ZFSN. Furthermore, the laws given in [Spivey, 1992] can be derived as theorems. Especially the theorem:

$$\bigcap_{x:\{\}} P(x) = \{y.\text{true}\}$$

holds, in contrast to ZF where the result of this intersection over the empty index set is defined equal to $\{\}$ because there are no universal sets in this untyped theory. In *typed* set theories like in Z or in HOL, the complement of a set is always defined.

5.2 Proof Support for the Schema Calculus

As discussed in the previous section, schemas can be used as predicates in the schema calculus, for which we implemented syntax and proof support. Besides the usual logical connectors $\wedge, \vee, \neg, \Rightarrow$ that may be used to connect schema expressions, there are also *schema-quantifiers* in the schema calculus. For example, the schema expression $\forall\, A \bullet B$ is a schema of type $\mathbb{P}(\langle\!|\, x_1' \mapsto \tau_1, x_2' \mapsto \tau_2 \,|\!\rangle)$. In HOL-Z, it is represented by:

$$\text{SB}\,\text{"}x_1'\text{"} \rightsquigarrow x_3\,\text{"}x_2'\text{"} \rightsquigarrow x_4 \bullet \forall (x_1, x_2) : \text{asSet}\ A \bullet B(x_1, x_2, x_3, x_4)$$

Analogously, the following operators are defined:

- the existential quantifier $\exists\, A \bullet B$,
- the *hiding* operator $B \setminus (x_1, x_2)$ equivalent to $\exists\, M \bullet B$ (where M is a schema with empty predicate part and schema-signature $[x_1, x_2]$), and
- the $\mathsf{pre}\, B$ operator that hides all variables that have a stroke or a "!"-suffix.

The latter schema operator is motivated by the convention in Z to give variables denoting components of a successor state a stroke suffix, while variables denoting output get a "!"-suffix.

Schema quantifiers play an important role for the formulation of proof obligations and lemmas in Z. The proof obligation Add_1 for the refinement in the BirthdayBook example (see Sec. 3.1) is a pure schema expression. For inserting local lemmas into Isabelle, proof goals can be inserted directly by a suitably adapted parser (using the Z-Encoder internally) based on the compact ASCII-based e-mail format defined in ZFSN. For example, one can insert an already simplified version of Add_1 described in [Spivey, 1992, p. 138][2]:

zgoal thy
"$\forall\ \mathit{BirthdayBook} \bullet \forall\ \mathit{BirthdayBook}1 \bullet \forall\, \text{n?} \in \mathit{NAME} \bullet \forall \text{d?} \in \mathit{DATE} \bullet$
    $(\text{n?} \notin \mathit{known} \wedge \ \mathit{known} = \{\text{n}.\ \exists\, i \in \#1..\mathit{hwm}.\ \text{n} = \mathit{names}\ i\}$
    $\Longrightarrow (\forall i \in \#1..\mathit{hwm}.\ \text{n?} \neq \mathit{names}\ i))$";

which opens an Isabelle proof state.

Note that this statement, directly drawn from a prominent Z textbook, is strictly speaking *not* a Z-formula in the sense of ZFSN; since there are logical connectors that have a schema expression on one side and a HOL expression on the other, such mixed expressions can not be entered in the ZeTa-frontend. Such use of mixed formulae in the course of proofs is in fact quite common in the Z literature (see also [Woodock and Davies, 1996]). Instead of developing a somewhat artificial, closed proof calculus

[2] For the purpose of our presentation we use the usual mathematical notation.

on schema expressions (as in [ISOZ, 2002; Henson and Reeves, 1998]), we opted for a calculus supporting such mixed forms.

The question has to be settled how the issue of binding is treated in mixed forms. Here, the general rule we adopted is that scopes introduced by schemas also extend to HOL subformulae, i.e. in $\forall S \bullet S'$, all bindings introduced by the schema-signature of S are also used to bind any free variables in S', regardless if it occurs in a schema expression or not. Moreover, the question has to be solved how schema expressions (i.e. expressions with a non-empty schema-signature) are treated logically at the top-level, since expression of the form are encoded into predicates over this schema-signature. Our answer is that we treat such "free variables" declared in a schema-signature but never bound as universally quantified; this is achieved by defining the $\vdash$ operator equivalent to the universal quantor $\forall$ of type $(\alpha \to \mathit{bool}) \to \mathit{bool}$ and adding it at the root of any schema expression. Conceptually, this means that $\vdash S$ (or: "valid S") has the meaning that the predicate must hold for all elements in the schema.

From the perspective of a mixed form calculus, it is quite clear what is needed for a joint calculus: for any construct of the schema calculus, a pair of introduction and elimination rules must be added. Since in the case of schema expressions, argument lists of predicates vary over schema-signatures, these rules are in fact rule schemes, whose individual instances are just equivalents for the usual (bounded) quantifiers and set comprehensions. In the following, we use $\mathbf{x}_i$ to denote a vector of variables $x_1, \ldots, x_n$; the juxtaposition $\mathbf{x}_i\mathbf{y}_i$ of two vectors represents their concatenation. With $\widetilde{\mathbf{x}}$ we denote a permutation of a vector $\mathbf{x}$. In the following, we represent the rule schemes of schema quantifiers in natural deduction style (to which Isabelle is mainly geared):

$$\frac{\bigwedge \mathbf{x}_i . S(\mathbf{x}_i)}{\vdash S}\ \text{turnstileI} \qquad\qquad \frac{\vdash S \qquad \begin{matrix}[S(\mathbf{t}_i)]\\ \vdots \\ R\end{matrix}}{R}\ \text{turnstileE}$$

$$\frac{\bigwedge \mathbf{x}_i .\ \begin{matrix}[S(\mathbf{x}_i)]\\ \vdots \\ T(\widetilde{\mathbf{x}_i\mathbf{y}_j})\end{matrix}}{\forall S \bullet T}\ \text{sch\_allI}(*) \qquad\qquad \frac{\forall S \bullet T \qquad \begin{matrix}[S(\mathbf{t}_i), T(\widetilde{\mathbf{t}_i\mathbf{y}_j})]\\ \vdots \\ R\end{matrix}}{R}\ \text{sch\_allE}(*)$$

$$\frac{S(\widetilde{\mathbf{y}_j\mathbf{t}'_i\mathbf{t}_k!})}{\mathit{pre}\ S}\ \text{preI}(*) \qquad\qquad \frac{\mathit{pre}\ S \qquad \bigwedge \mathbf{x}'_i.\mathbf{x}_k!.\ \begin{matrix}[S(\widetilde{\mathbf{y}_j\mathbf{x}'_i\mathbf{x}_k!})]\\ \vdots \\ R\end{matrix}}{R}\ \text{preE}(*)$$

$$\frac{S(\mathbf{t}_i) \quad S(\widetilde{\mathbf{t}_i\mathbf{y}_j})}{\exists S \bullet T}\ \text{sch\_exI}(*) \qquad\qquad \frac{\exists S \bullet T \qquad \bigwedge \mathbf{x}_i .\ \begin{matrix}[S(\widetilde{\mathbf{x}_i\mathbf{y}_j})]\\ \vdots \\ R\end{matrix}}{R}\ \text{sch\_exE}(*)$$

Note that the *turnstileI* rule introduces the equivalent of fresh free variables into a backward proof-state; consequently, schema expressions are not necessarily closed in our calculus. This motivates the following proviso (*) on most of the rules above: we require that $\mathbf{y}_k$ is the vector of free variables corresponding to the schema signature of the conclusion P (in the introduction rules) or the type of the first premise P (in the elimination rules); i.e. we require $\mathbf{y}_k = S_\Sigma(\tau)$ where τ is the type of P.

In HOL-Z, for each of these rule schemes a special tactic is provided for both forward and backward proof. While the former corresponds to a transformation on objects of type *thm* — representing formulae accepted by Isabelle as valid —, the latter is a `tactic` that may be applied to the i-th subgoal of the current proof state. These tactics are collected in the SML package `ZProofUtils` and have the format:

```
val strip_turnstile : thm → thm
(* erases topmost turnstile ⊢ S *)

val strip_schball : thm → thm
(* erases topmost schema quantifier ∀ x: A • P x *)

val intro_sch_all_tac : int → tactic
(* pseudo introduction rule of a schema-universal quantifier
   ∀ S • P; in backwards reasoning, it eliminates a topmost
   schema-quantifier and replaces them by parameters, that
   were suitably renamed *)

val elimsch_all_tac : int → tactic
(* pseudo elimination rule of a schema-universal quantifier
   ∀ S • P; in backwards reasoning, it eliminates a topmost
   schema-quantifier in the assumption list and replaces them
   by schema variables. *)
```

An introduction and elimination rule pair for schema comprehensions $\{S \mid P \bullet E\}$ (semantically represented as $\{m \mid \exists \mathbf{x}_i : \text{asSet}S \bullet P(\mathbf{x}_i) \wedge m = E(\mathbf{x}_i)\}$) is also provided. Its definition is straightforward and not really a semantic extension of HOL, merely a syntactic paraphrasing of HOL rules.

While the correctness of the calculus is assured by formal, machine-checked proofs of more atomic rules that were used inside the tactics implementing the above rule schemes, the question of (relative) completeness is more difficult to answer. Of course, since HOL includes the axiom of infinity, HOL is incomplete wrt. standard models as a consequence of Gödels incompleteness results. However, from the form of the rule schemes, it is obvious that they "transform" *all* schema expressions into standard higher-order predicate expressions in the course of a proof; this means, that for monomorphic expressions, the completeness result [Andrews, 1986, p. 197] applies here wrt. *Henkin models* and a calculus presented there. To the best of our knowledge, there is no (relative) completeness result for the polymorphic case and the precise form of rules used in Isabelle/HOL.

5.3 Interfacing Schema Expressions into Proof-Contexts

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. These operators are available both in a forward and backward version and are declared as:

```
val stripS : thm → thm
(* erases topmost combination of operators as above *)

val stripS_tac : int → tactic
(* generalization of HOL's strip_tac - removes leading
   turnstiles, universal schema, bounded and unbounded
   quantifiers and implications ... *)
```

These tactical operations serve as proof-technical adaptors between Z-style lemma formats and a presentation in terms of the built-in logic `Pure` of Isabelle. If one thinks of a schema-signature as an interface to the parameters of its context proof state, both the forward and backward combinators work as kind of interface adaptors: in a backward proof, `stripS_tac` opens the local bindings hidden internally in the schema quantifiers by converting them to a parameter context, i.e. a vector of variables bound by Isabelle's meta-logic quantifier $\bigwedge$ (traditionally used to implement provisos in logical rules of the form "this variable must not occur in the assumptions", etc.). Conversely, a Z-style lemma may be logically "massaged" via `stripS` before introducing it into a backward proof; such a massage consists in erasing all schema binders and replacing bound variables in the formula by meta-variables that may be instantiated by the parameters of the proof context by Isabelle's resolution. We would like to emphasize again that all these highly non-trivial transformations on the binding structure in a mixed form of HOL-Z are based on rules derived from the definitions of the schema-logical quantifiers and thus proven correct within Isabelle. The applications of these elementary rules are controlled by tactical programs, that also apply elementary renaming tactics to present the bound variables of the proof-state in terms of user-defined names stemming from the specification.

5.4 Semantic Projections "on the fly"

From a pragmatic point of view, schemas represent nested containers of semantic knowledge of the specification. Experience shows that just expanding schemas of realistic size in the course of a proof is usually infeasible; proof states tend to become too large to be accessible to both interactive and automatic reasoning. For this, in the course of a proof, expansions of schemas should be avoided. Rather, if a particular consequence of a schema is needed, a *semantical projection lemma* should be used; for our example schema B (see Fig. 3), these are the following lemmas in mixed form:

$$\vdash B \Longrightarrow A \qquad \vdash B \Longrightarrow x_2 \in T \qquad \vdash B \Longrightarrow A' \qquad \vdash B \Longrightarrow Q$$

We provide special functions that generate semantic projections "on the fly" whenever they are needed:

```
val get_decl : theory → string → int → thm
val get_conj : theory → string → int → thm
```

The first lemma in the list of semantic projection lemmas for B can be generated by `get_decl thy "B" 1`, while the last is constructed by `get_conj thy "B" 1` ("give the first conjunct of the predicate part of schema B). Via the `stripS`-combinator, semantic projection lemmas can be converted into Isabelle's meta-logical format "on the fly" and therefore be used in a backward proof (see the example in the next section).

5.5 An Example for Structured Proofs in HOL-Z

In order to demonstrate the proof techniques introduced in the previous sections, we use a standard textbook proof [Spivey, 1992, p. 138] for the first proof obligation of the refinement of *AddBirthday* by *AddBirthday*1. Spivey argues that this theorem can be immediately reduced to the following simplified form:

zgoal thy
"$\forall$ *BirthdayBook*$\bullet$ $\forall$ *BirthdayBook*1 $\bullet$ $\forall$ n? $\in$ *NAME*$\bullet\forall$d? $\in$ *DATE*$\bullet$
(n? $\notin$ *known*$\wedge$ *known*= {n. $\exists\, i{\in}\#1..hwm.$ n=*names* i}
$\Longrightarrow (\forall i \in \#1..hwm.$ n? $\neq$ *names* i))";

and the application of

by(stripS_tac 1);

transforms the goal into the following proof state:

1. $\bigwedge$*birthday known dates hwm names* n? d? *i*.
   $\llbracket$ *BirthdayBook*(*birthday*, *known*); *BirthdayBook*1 (*dates*, *hwm*, *names*);
   n? $\in$ *NAME*; d? $\in$*DATE*;
   n? $\notin$ *known*$\wedge$ *known*= {n. $\exists$ *i*$\in$#1..*hwm*. n=*names i*};
   i $\in$ (#1 .. *hwm*) $\rrbracket$ $\Longrightarrow$n? $\neq$ *names i*

Note that the quite substantial reconstruction of the underlying binding structure still leads to a proof state that is similar in style and presentation to [Woodock and Davies, 1996].

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

auto();

which finishes the proof automatically.

Unfortunately, a more careful analysis of the initial proof obligation Add_1 (Sec. 3.1) and the "simplified" formulation above represents a gap in Spivey's proof. The implicitly assumed `lemma1`:

$\vdash$*BirthdayBook*$\wedge$ ($\forall$*i*$\in$#1..*hwm*. n? $\neq$*names i*) $\Longrightarrow$pre AddBirthday1

states that a valid concrete state *BirthdayBook*1 and the *syntactic precondition* (i.e. the conjoints in the predicate part of a schema that contain only occurrences of variables without stroke or with a "?"-suffix) implies the *semantic precondition* (i.e. *pre S* meaning "there is a successor state"). In other words, any reachable state $\lbag$ $names' == a, hwm' == b, dates' == c$ $\rbag$ fulfills the state invariant *BirthdayBook*$'$, i.e. $\forall$i,j$\in$#1..*hwm*'. i$\neq$j $\Longrightarrow$*names*'(i)$\neq$*names*'(j). This proof constitutes in fact 80 percent of the overall proof task and is omitted here (see our example documentation in the HOL-Z 2.0 distribution).

Instead, we focus on a sample proof that shows how the bits and pieces can be brought together: We start the proof with the generated proof obligation Add_1; its formula is bound to a constant that is unfolded during the initialization of the proof:

{ goalw thy [Add_1def] "Add_1";}

$\forall$ *BirthdayBook*$\bullet$ ($\forall$ *BirthdayBook*1 $\bullet$($\forall$ n?$\in$*NAME*. $\forall$d?$\in$*DATE*. pre AddBirthday $\wedge$Abs $\Longrightarrow$pre AddBirthday1)))

The next steps represent the opening of the bindings and some structural normalization:

$\Longleftarrow\left\{\begin{array}{l}\text{by (stripS\_tac 1);}\\ \text{by (Step\_tac 1);}\end{array}\right\}$

$\forall$*birthday known dates hwm names* n? d?.
⟦ *BirthdayBook*(*birthday*, *known*); *BirthdayBook*1 (*dates*, *hwm*, *names*);
n? $\in$ *NAME*; d? $\in$*DATE*;
pre (AddBirthday(*birthday*, *birthday*', d?, *known*, *known*, n?));
Abs (*birthday*, *dates* , *hwm*, *known*, *names*) ⟧
$\Longrightarrow$ pre AddBirthday1

We apply `lemma1` and eliminate its premise `BirthdayBook` from the proof context:

$$\Longleftarrow \left\{ \begin{array}{l} \text{by (rtac (stripS lemma1) 1);} \\ \text{br conjI 1;} \\ \text{by (convert2hol\_tac [] 1);} \end{array} \right\}$$

$\forall$*birthday known dates hwm names* n? d?.
⟦ *BirthdayBook*(*birthday*, *known*); *BirthdayBook*1 (*dates*, *hwm*, *names*);
n? $\in$ *NAME*; d? $\in$*DATE*;
pre (AddBirthday(*birthday*, *birthday*', d?, *known*, *known*, n?));
Abs (*birthday*, *dates* , *hwm*, *known*, *names*) ⟧
$\Longrightarrow \forall$ i$\in$ #1 .. *hwm*. n? $\neq$ *names* i

Now we weaken the assumptions by applying `lemma2` (this simple lemma can be found in the HOL-Z 2.0 distribution and is not explained here), and by applying a semantic projector into schema `Abs` yielding its first conjunct:

$$\Longleftarrow \left\{ \begin{array}{l} \text{bd (stripS lemma2) 1;} \\ \text{bd (stripS (get\_conj thy “Abs” 1)) 1;} \end{array} \right\}$$

$\bigwedge$*birthday known dates hwm names* n? d?.
⟦ *BirthdayBook*(*birthday*, *known*); *BirthdayBook*1 (*dates*, *hwm*, *names*);
n? $\in$ *NAME*; d? $\in$*DATE*; n? $\notin$*known*;
known= *known*= {n. $\exists$ *i*$\in$#1..*hwm*. n=*names* i} ⟧
$\Longrightarrow \forall$ i$\in$ #1 .. *hwm*. n? $\neq$ *names* i

We are now in the position described before in Spivey's simplified proof, such that Isabelle's standard proof procedure can take over and complete the proof:

$\Longleftarrow$ {auto();}

True

This closes our example proof. For the sake of the presentation, we deliberately chose the procedural proof-language of Isabelle and not the more recent, declarative one called *Isar*. We consider an integration into *Isar* as an add-on that complicates matters here. In any case, an integration into Isar would be a useful extension of the actual HOL-Z environment.

6 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 write our Z specifications in the type setting system LaTeX, we can 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, becomes feasible.

We applied HOL-Z to several large specifications, e.g. an architecture of CVS (the Concurrent Versions System) [Brucker et al., 2002] and the CORBA Security Service [Basin et al., 2002], with a focus on security analysis of CVS and CORBA. The large CORBA example (approx. 90 pages (!) that are converted and loaded in less than 5 minutes on a standard PC using PolyML) shows the feasibility of our approach for real world examples. The case studies also involved significant proofs of the refinement of an abstract architectural description to the implementation.

A consequence of our implementation of the converter is that there is no direct interaction 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 would be, to our knowledge, a new technique for automated deduction on Z specifications.

References

[**Andrews, 1986**] Andrews, P. B. (1986). *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press.

[**Basin et al., 2002**] Basin, D., Rittinger, F., and Viganò, L. (2002). A formal analysis of the CORBA security service. In Bert, D., Bowen, J. P., Henson, M. C., and Robinson, K., editors, *ZB 2002: Formal Specification and Development in Z and B*, LNCS 2272, pages 330–349. Springer.

[**Bowen and Gordon, 1995**] Bowen, J. P. and Gordon, M. J. C. (1995). A shallow embedding of Z in HOL. *Information and Software Technology*, 37(5–6):269–276.

[**Brucker et al., 2002**] Brucker, A. D., Rittinger, F., and Wolff, B. (2002). A CVS-Server security architecture — concepts and formal analysis. Technical Report 182, Albert-Ludwigs-Universität Freiburg.

[**Brucker and Wolff, 2002**] Brucker, A. D. and Wolff, B. (2002). A proposal for a formal OCL semantics in Isabelle/HOL. In Muñoz, C., Tahar, S., and Carreño, V., editors, *Theorem Proving in Higher Order Logics*, LNCS 2410, pages 99–114. Springer.

[**Church, 1940**] Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68.

[**FDR, 2003**] FDR (2003). Failures-divergence refinement – FDR2 user manual. `http://www.fsel.com/fdr2_manual.html`.

[**Gordon and Melham, 1993**] Gordon, M. J. C. and Melham, T. F. (1993). *Introduction to HOL.* Cambridge University Press.

[**Henson and Reeves, 1998**] Henson, M. C. and Reeves, S. (1998). A logic for the schema calculus. In Bowen, J. P., Fett, A., and Hinchey, M. G., editors, *ZUM'98: The Z Formal Specification Notation*, LNCS 1493, pages 172–191. Springer.

[**ISOZ, 2002**] ISOZ (2002). Z formal specification notation — syntax, type system and semantics. ISO/IEC 13568:2002, International Standard.

[**KIV, 2003**] KIV (2003). `http://i11www.ira.uka.de/~kiv/`.

[**Kolyang et al., 1996**] Kolyang, Santen, T., and Wolff, B. (1996). A structure preserving encoding of Z in Isabelle/HOL. In von Wright, J., Grundy, J., and Harrison, J., editors, *Theorem Proving in Higher Order Logics*, LNCS 1125, pages 283–298. Springer Verlag.

[**Naraschewski and Wenzel, 1998**] Naraschewski, W. and Wenzel, M. (1998). Object-oriented verification based on record subtyping in Higher-Order Logic. In Grundy, J. and Newey, M., editors, *Theorem Proving in Higher Order Logics*, LNCS 1479, pages 349–366. Springer.

[**Paulson, 1994**] Paulson, L. C. (1994). *Isabelle: a generic theorem prover.* LNCS 828. Springer, New York.

[**Paulson, 2000**] Paulson, L. C. (2000). Mechanizing UNITY in Isabelle. *ACM Transaction on Computational Logic*, 1(1):3–32.

[**Reetz, 1995**] Reetz, R. (1995). Deep Embedding VHDL. In Schubert, E., Windley, P., and Alves-Foss, J., editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, LNCS 971, pages 277–292. Springer.

[**Santen, 1998**] Santen, T. (1998). On the semantic relation of Z and HOL. In Bowen, J., Fett, A., and Hinchey, M., editors, *ZUM '98*, LNCS 1493, pages 96–115.

[**Spivey, 1992**] Spivey, J. M. (1992). *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science.

[**Tej and Wolff, 1997**] Tej, H. and Wolff, B. (1997). A corrected failure-divergence model for CSP in Isabelle/HOL. In Fitzgerald, J., Jones, C., and Lucas, P., editors, *FME 97*, LNCS 1313, pages 318–337. Springer.

[**Woodock and Davies, 1996**] Woodock, J. and Davies, J. (1996). *Using Z.* Prentice Hall.

[**Zeta, 2003**] Zeta (2003). `http://uebb.cs.tu-berlin.de/zeta/`.

[**ZEVES, 2003**] ZEVES (2003). `http://www.ora.on.ca/z-eves/welcome.html`.