A Set-Based Calculus and its Implementation

vorgelegt von Diplom-Informatiker
Wolfgang Grieskamp
aus Essen

Am Fachbereich 13, Informatik,
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzender	:	Prof. Dr. Günter Hommel
Berichter	:	Prof. Dr. Peter Pepper
Berichter	:	Prof. Dr. Stefan Jähnichen

Tag der wissenschaftlichen Aussprache: 26.11.1999

Berlin 1999

D 83

Zusammenfassung

Diese Arbeit untersucht die *Fundierung* und *Implementierung* eines *Berechnungsmodells* für *mengenbasierte* Computersprachen. Es wird das μZ-Kalkül eingeführt, eine minimale mengenbasierte Ausdruckssprache, verwandt mit dem λ-Kalkül, das Anwendungssprachen wie die Spezifikationssprache Z oder funktional-logische Programmiersprachen komplett und natürlich einbetten kann. *Syntax*, *Semantik* und *Gleichungstheorie* von μZ werden definiert. Mehrere *Berechnungsmodelle* werden entwickelt. Durch gezielte Anwendung der Gleichungstheorie ergibt sich eine *Narrowingsemantik*, die durch lokale Modifikation der Redexauswahl nicht-strikt oder strikt ausfällt, und insbesondere interessant für interpretative "on-line" Auswertung ist, da sie auf symbolischer Termersetzung basiert. Das zentrale Berechnungsmodell aber basiert auf einer *natürlichen Semantik*, die durch ihren operationalen Charakter das Design kompilierter "off-line" Auswertung vorbereitet, und das *Berechnungsreferenzmodell* von μZ festlegt. Dieses Modell kombiniert funktionale Reduktion und nebenläufige Resolution, und definiert eine klare Trennung zwischen der (deterministischen, funktionalen) Konstruktion von Mengen und der (nicht-deterministischem, logischen) Dekonstruktion. Da Mengen sowohl extensional als auch intensional dargestellt werden können, ist das Modell von höherer Ordnung.

Die Implementierung erfolgt in Form einer abstrakten Maschine, der ZAM, welche auf der Technik nebenläufiger Resolutionsagenten basiert. Die Maschine realisiert eine effiziente Tiefensuche über die Verwaltung von Auswahlpunkten und Modifikationsprotokollen. Durch ein Prioritätenmodell in der nebenläufigen Ausführung werden nichtdeterministische Berechnungen dynamisch verzögert und deterministische propagiert. Die ZAM ist als Stackmaschine vollständig in Z spezifiert, und wird in der Implementierung in C++ zu einer Registermaschine verfeinert, welche *persistente* Register verwendet, um den Aufwand der Protokollierung von Modifikation zu minimieren.

Die Anwendung der vorgestellten Konzepte erfolgt im Rahmen des ZeTa Systems, im BMBF Projekt ESPRESS entwickelt, das verschiedenartige Notationen und Werkzeuge auf der semantischen Grundlage von Z integriert. Der ZaP Übersetzer, Bestandteil des ZeTa Systems, realisiert mit Hilfe des μZ-Kalküls die Ausführung von Z Modellen, und wurde beispielsweise in ESPRESS für die Automatisierung von Softwaretests eingesetzt.

Die Definition von μZ, der Berechnungsmodelle, der abstrakten Maschine und der Übersetzung von μZ in Maschinencode ist vollständig in der Spezifikationssprache Z gegeben (mit kleineren syntaktischen Erweiterungen), und daher syntax- und typgeprüft. Insofern ist diese Arbeit auch eine Fallstudie der Anwendbarkeit von Z für anspruchsvolle Probleme der Metamodellierung.

Contents

1 Introduction **17**

2 Meta Notation **23**
2.1 A Sketch of Z . . . 25
2.1.1 Types and Sematic Universe . . . 25
2.1.2 Schema Text . . . 25
2.1.3 Genericity . . . 26
2.1.4 Undefinedness . . . 26
2.2 Notational Extensions . . . 28
2.2.1 Where Notation . . . 28
2.2.2 If-Exists Notation . . . 28
2.2.3 Inference Rule Systems . . . 28
2.2.4 Recursive Partial Functions . . . 29
2.2.5 Local Variables in Schema Boxes . . . 29
2.2.6 Case Distinction in Predicates . . . 30
2.2.7 Selective Δ . . . 30
2.2.8 Overloading . . . 30
2.2.9 Further Extensions . . . 31
2.3 Mathematical Tools . . . 32
2.3.1 Sets and Relations . . . 32
2.3.2 Sequences . . . 32
2.3.3 Booleans . . . 33
2.3.4 Order and Lattices . . . 33

3 The μZ Calculus **37**
3.1 Introduction to μZ . . . 38
3.1.1 Syntax and Informal Meaning . . . 38
3.1.2 Encodings . . . 39
3.2 Semantic Model . . . 43
3.2.1 Set Representation . . . 43
3.2.2 Types . . . 44
3.2.3 Universe . . . 45
3.2.4 Free Types . . . 46
3.2.5 Standard Types and Constructors . . . 47
3.2.6 Order and Fixed-Points . . . 48
3.3 Expression Syntax and Meaning . . . 50

3.3.1 Variable, Expression and Pattern Type 50
3.3.2 Typing Relation 50
3.3.3 Meaning Function 51
3.3.4 Basic Forms: Typing and Meaning 52
3.3.5 Expression Abbreviation Forms 56
3.4 Equational Theory 57
3.4.1 Syntactic Tools 57
3.4.2 Equivalence Relation 57
3.4.3 Boolean Laws 58
3.4.4 Singleton Set Laws 59
3.4.5 Translation Laws 60
3.4.6 Schema Laws 64
3.4.7 Fixed-Point Unrolling Law 66
3.5 Discussion 67

4 Computation in μZ 71
4.1 Preliminaries 72
4.2 Normalization 74
4.2.1 Property Normalization 75
4.2.2 Normalization Rules 76
4.2.3 Properties of Normalization 79
4.3 Narrowing Semantics 82
4.3.1 General Narrowing 82
4.3.2 Outermost-in Narrowing 84
4.3.3 Strict Narrowing 85
4.4 The Reference Computation Model 86
4.4.1 Values and Constraints 87
4.4.2 Computation 88
4.5 Discussion 93

5 Abstract Machine 97
5.1 Basic Model of the ZAM 98
5.1.1 Values 98
5.1.2 Intensions 99
5.1.3 Threads and Goals 100
5.1.4 Configurations 102
5.1.5 Instructions 102
5.2 Specification 105
5.2.1 Axiomatic Definitions 105
5.2.2 Auxiliary Instructions 107
5.2.3 Execution Step and Selection 109
5.2.4 Instructions 110
5.3 Compilation 124
5.3.1 Environment 124
5.3.2 Compilation Functions 125
5.4 Implementation 129

5.4.1 Data Model 129
5.4.2 Instruction Set 131
5.4.3 Memory Management 133
5.4.4 Performance 133
5.5 Discussion 135

6 The ZeTa System 139
6.1 Three Dimensions of Integration 139
6.1.1 Semantic Integration 140
6.1.2 Document Integration 141
6.1.3 Tool Integration 141
6.1.4 Graphical User Interfaces 143
6.2 The ZaP Compiler 144
6.2.1 The Basic Specification of the Birthday Book 146
6.2.2 Refining the Birthday Book for Execution 147
6.2.3 Refining the Birthday Book for Testing 150
6.3 Discussion 154

7 Conclusion 157
7.1 Contributions 157
7.2 Future Work 159

Ich ging im Walde
So für mich hin,
Und nichts zu suchen,
Das war mein Sinn.
(Goethe)
[1]

Chapter 1

Introduction

The automatic evaluation of *test data* for *safety-critical embedded systems* is an interesting application that can help to put declarative language implementation techniques into industrial practice. Some studies report that more than 50% of development costs in this application area go into testing. A setting for test-case evaluation that can improve this situation is as follows: given a requirements specification by an executable prototype, some input data describing a test case, and the output data from a run of the system's implementation on the given input, we check by executing the specification whether the implementation meets its requirements. At the first sight, this goal would seem to be simple, since input and output data are fixed. However, a real-world specification of a complex system may contain a lot of "hidden" data, which is used to describe the observable behavior. Thus, the problem scales up to finding the solution(s) to (a sequence of) partial data bindings.

In the application-oriented research project ESPRESS[2], which is concerned with methods and tools for the development of embedded systems, the set-based specification language Z [Spivey, 1992] is used for requirements specification in combination with other notations such as Statecharts which are incorporated by a shallow encoding in Z [Büssow and Grieskamp, 1999]. Performing test-case evaluation in this setting requires a *computation model* for Z.

In this thesis, the underlying *theory* and *implementation* of a computation model for Z are investigated. In fact, the work is not restricted to Z, since it is based on a small intermediate calculus, called μZ, that can embed Z as well as other declarative notations. The major contributions are the introduction of the novel calculus μZ, the definition of the calculus' computation model (operational semantics) and the definition of an abstract machine for implementing the calculus. In this initiatory Chapter, an outline of these topics is given.

[1] Im Goethejahr 1999 haben wir uns vorgenommen, (fast) alle Kapitelwahlsprüche beim großen deutschen Dichter auszuleihen.

[2] ESPRESS is a joint project involving industrial partners (Daimler-Benz AG and Robert Bosch AG) and academic research institutes (GMD-First, FhG-ISST, and TU-Berlin), funded by the German "Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie".

The μZ Calculus

The μZ calculus is a pure expression language in the spirit of the λ-calculus. Since it can embed the full Z language – the simple-typed λ-calculus, set calculus, predicate calculus, and schema calculus – computation in μZ can not be complete. However, a computation model and implementation by an abstract machine can be provided that is comparable with that of higher-order functional logic paradigms [Smolka, 1998; Hanus, 1999; Chakravarty et al., 1997].

The μZ calculus is based upon a small number of constructs, which are tentatively summarized below ($e \in EXP$ being expressions and $p \in PAT$ patterns):

$$\begin{array}{l} e \in EXP ::= x \mid \rho(\overline{e}) \mid \{e\} \mid \mu\, e \mid 0 \mid e \cap e' \mid e \cup e' \mid \sim e \mid \\ \qquad\qquad e[p_1 \mapsto p_2] \mid \{p \mid e\} \mid \mathbf{fix}\, p \lhd e \\ p \in PAT \subseteq EXP \end{array}$$

The calculus employs constructor terms, $\rho(\overline{e})$, and the basic operations of set algebra, set union, intersection and complement. Patterns, $p \in PAT$, are the subset of expressions built from constructor terms and variables. In addition, the following expression forms are provided:

- The form $\mu\, e$ selects the element of a singleton set e.
- The form $e[p_1 \mapsto p_2]$ describes the *translation* of the set e. The result consists of all elements that match p_2, such that there exists an element in e that matches the pattern p_1 under the same substitution. For example, $(\{\rho(1,2)\} \cup \{\rho(1,3)\})[\rho(x,y) \mapsto x] \equiv \{1\}$ (where numbers are special 0-ary constructor symbols).
- The form $\{p \mid e\}$ describes a *schema* (set comprehension). The property e is a set expression of Boolean type – truth being the set containing a dedicated singleton element and falsity the empty set.
- The form $\mathbf{fix}\, p \lhd e$ describes a fixed-point of a set (or tuple of sets) e.

The calculus is able to represent all concepts of Z in a remarkably consistent way. For example, a Cartesian product $e_1 \times e_2$ is expressed as $e_1[x \mapsto (x, _)] \cap e_2[y \mapsto (_, y)]$, with $(_, _)$, the constructor symbol for pairs, used in mixfix notation, and $_$ a pattern "joker" (or anonymous variable). Relational application, $R\, e$, is encoded as $(R \cap \{e\})[_ \mapsto \diamond]$, where $\diamond$ is the 0-ary constructor for the unit in Boolean sets. Function application, $e_1\, e_2$, is given as $\mu((e_1 \cap \{e_2\}[x \mapsto (x, _)])[(_, y) \mapsto y])$. As these examples indicate, the heart of the μZ calculus is its novel construct for set translation, which provides a means for the constructive description of morphisms in the category of μZ sets.

In Chapter 3, we develop the exact syntax and semantics of the calculus and provide an equational theory for it. The semantic model of μZ is based on a hierarchical typed universe, where sets are represented as *partial characteristic (Boolean) functions* in a set-theoretical sense, which can be founded by Z itself or by ZF set theory. This representation of sets is a generalization of partial three-valued logics [Owe, 1997] to the case of set algebra. The equational theory of μZ is powerful enough to induce a *narrowing semantics* by a pure expression transformation, as will be shown in Chapter 4.

Computation in μZ

Several notions of computation are investigated in Chapter 4. We start with a *normalization* of expressions to a disjunctive normal form, which, combined with simplification, is a prerequisite for all further investigations (including compilation). From the normalization, we can easily deduce a general *narrowing semantics* which is suited for symbolic "on-line" computation. By restricting the redex selection order of narrowing, several submodels are derived.

The reference computation model of μZ, however, is given in the style of *natural semantics*, constituting a trade-off between computation power and efficient implementability, preparing a straight transition to the abstract machine given in Chapter 5 for "off-line" computation. This model draws a clear borderline between the *functional* (deterministic) construction of sets and the *logic* (indeterministic) deconstruction. For example, in an expression such as $\{x_1 \mid x_1 \in e_1\} \cap \{x_2 \mid x_2 \in e_2\}$ (where $_ \in _$ is syntactic sugar), the expressions e_1 and e_2 are concurrently reduced in a deterministic functional style. Once an expression e_i has reached a value form, a resolution step is performed, thereby binding the variable x_i.

The reference computation model of μZ is comparable with that of higher-order functional logic languages with concurrent constraint resolution, though it goes beyond this, supporting disjunction and negation. It supports the free combination of sets (relations and functions) by set union, intersection and complement, thereby abstracting from intensional and extensional representations. The widely used set-based Z specification style can thus be fully supported. We give some examples in Z. Given the function f, the execution of its domain restriction, $\{e_1, \ldots, e_n\} \lhd f$, causes no problems. We can also execute the properties $x \in \mathsf{dom}\, f$ or $x \in \mathsf{ran}\, f$. We may execute $f^{\sim}$, which might become a relation, if f is not injective. If g is a further function, then $f \cup g$ and $f \cap g$ are executable. Similar possibilities carry over to the schema calculus, which can be fully executed as well.

The limitations of the computation model are found in the executability of universal quantification and of complement. The universal quantification, $\forall x \bullet e$, is encoded in μZ as $\sim (\{x \mid \sim e\}[x \mapsto \diamond])$. The problem of executing this construct is conspicuously apparent in the equational theory of μZ: set complement cannot be distributed over hiding translation. Thus, we cannot expect complements to be pushed down completely to the expression leafs during the construction of the disjunctive normal form. We deal with this problem by computing independent "subgoals" for expressions nested inside of complements. These computations do not contribute to resolution in their enclosing context, and are bound to the context by residuation. For the universal quantifier, this entails that it can be computed provided the quantified range is finitely enumeratable.

The Abstract Machine ZAM

An abstract machine – called ZAM – which implements the reference computation model of μZ is developed in Chapter 5. In the ZAM, concurrent threads collaboratively work on a resolution task. The threads synchronize via variables: accessing an unbound variable suspends a thread and binding a variable resumes threads waiting for it (residuation). The machine maintains a stack of choice points which is shared by the threads jointly acting on a resolution. When a thread executes a property such as $x \in e_1 \cup e_2$, it continues

with $x \in e_1$ and creates a choice point for $x \in e_2$. A simple but powerful optimization is achieved by an instance of the "Andorra Principle" [Warren, 1990], dynamically lowering the priority of threads that try to create choice points, thereby giving deterministic computation preference.

The compilation of μZ to ZAM instructions is based on the disjunctive normal form defined in Chapter 4. The instruction set of the ZAM is at a higher level than, for example, of the Warren Abstract Machine [Warren, 1983], making compilation easy and retraceable.

In Chapter 5, we give a comprehensive specification of the ZAM in Z's sequential specification style. This specification is intended to be executable using the concepts presented in this thesis. A prototype of the ZAM has been implemented in C++, and will be also outlined.

The ZeTa System and Conclusion

In Chapter 6, we give a short overview of the design of the ZeTa system [Büssow et al., 1998; Büssow and Grieskamp, 1999], and the role of the ZAP compiler as integrated into ZeTa. The goal of the ZeTa system is to offer an *open* environment for editing, browsing and analyzing integrated specifications assembled from heterogeneous formalisms – such as Statecharts, Z, temporal logic, message sequence charts, and other. The formalisms are semantically integrated by a shallow encoding in Z, using the $\mu\mathcal{SZ}$ conventions [Büssow et al., 1997a]. The ZeTa system is designed to systematically support such encodings by three dimensions of integration: semantic integration, document integration and tool integration. The ZAP compiler, part of the ZeTa system, can thus be used not only to execute Z specifications, but also other formalisms that are mapped to Z and thereby to μZ.

Chapter 7 concludes with a summary of the contributions of this work and of future lines of research. A discussion of related work can be found at the end of each of the relevant chapters.

Reading Advices

Since this thesis aims at a comprehensive formalization of the concepts it introduces, it contains a lot of detailed formal material, most of which as been put into figures that are interleaved with the text. For layout technical reasons, figures occasionally occur before their contents is explained. It is recommended to read (at least) until the first reference to a figure in the text until its content is read.

Not all parts of this thesis need to be thoroughly read to get an idea of the main concepts. Chapter 2 may be skipped by readers which are firm to Z and which do not want to know how we explain certain Z extensions that have common roots in well-known notational concepts (such as inference rules). In Chapter 3, it might be sufficient to read the introduction to the μZ calculus, Section 3.1 (on page 38), then the equational theory, Section 3.4 (on page 57), skipping the proofs in there, and finally the discussion. In Chapter 4, the definition of the distributive normal form in the introduction of Section 4.2 (on page 74) and the entire Section 4.4 (on page 86) is essential, and, of course, the discussion. In Chapter 5, the reader might find it sufficient to read Section 5.1 (on page 98),

Section 5.4 (on page 129) and the discussion. Chapter 6 may be skipped by readers not interested in the application.