Formal Specification and Verification Techniques

Prof. Dr. K. Madlener

21. Oktober 2010

Course of Studies „Informatics“, „Applied Informatics“ and „Master-Inf.“ WS10/11
Prof. Dr. Madlener
TU- Kaiserslautern

Lecture:

Mo 08.15–09.45	48-462	We 08.15–09.45	48-462

Exercises:??

Fr. 11.45–13.15	32-439	Mo 13.45–15.45	32-439

- Information `http://www-madlener.informatik.uni-kl.de/teaching/ss2009/fsvt/fsvt.html`
- Evaluation method:
  Exercises (efficiency statement) + Final Exam (Credits)
- First final exam: (Written or Oral)
- Exercises (Dates and Registration): See WWW-Site

Bibliography

M. O'Donnell.
Computing in Systems described by Equations, LNCS 58, 1977.
Equational Logic as a Programming language.

J. Avenhaus.
Reduktionssysteme, (Skript), Springer 1995.

Cohen et.al.
The Specification of Complex Systems.

Bergstra et.al.
Algebraic Specification.

Barendregt.
Functional Programming and Lambda Calculus. Handbook of TCS, 321-363, 1990.

Bibliography

- Gehani et.al.
  Software Specification Techniques.
- Huet.
  Confluent Reductions: Abstract Properties and Applications to TRS, JACM, 27, 1980.
- Nivat, Reynolds.
  Algebraic Methods in Semantics.
- Loeckx, Ehrich, Wolf.
  Specification of Abstract Data Types, Wyley-Teubner, 1996.
- J.W. Klop.
  Term Rewriting System. Handbook of Logic, INCS, Vol. 2, Abransky, Gabbay, Maibaum.

Bibliography

- Ehrig, Mahr.
  Fundamentals of Algebraic Specification.
- Peyton-Jones.
  The Implementation of Functional Programming Language.
- Plasmeister, Eekelen.
  Functional Programming and Parallel Graph Rewriting.
- Astesiano, Kreowski, Krieg-Brückner.
  Algebraic Foundations of Systems Specification (IFIP).
- N. Nissanke.
  Formal Specification Techniques and Applications (Z, VDM, algebraic), Springer 1999.

Bibliography

- Turner, McCluskey.
  The construction of formal specifications. (Model based (VDM) + Algebraic (OBJ)).
- Goguen, Malcom.
  Algebraic Semantics of Imperative Programs.
- H. Dörr.
  Efficient Graph Rewriting and its Implementation.
- B. Potter, J. Sinclair, D. Till.
  An introduction to Formal Specification and Z. Prentice Hall, 1996.

Bibliography

J. Woodcok, J. Davis.
Using Z: Specification, Refinement and Proof, Prentice Hall 1996.

J.R. Abrial.
The B-Book; Assigning Programs to Meanings. Cambridge U. Press, 1996.

E. Börger, R. Stärk
Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, 2003.

F. Baader, T. Nipkow
Term Rewriting and All That. Cambridge, 1999.

Goals - Contents

General Goals:

Formal foundations of Methods
for Specification, Verification and Implementation

Summary

- The Role of formal Specifications
- Abstract State Machines: ASM-Specification methods
- Algebraic Specification, Equational Systems
- Reduction systems, Term Rewriting Systems
- Equational - Calculus and - Programming
- Related Calculi: λ-Calculus, Combinator- Calculus
- Implementation, Reduction Strategies, Graph Rewriting

Lecture's Contents

Abstract State Machines (ASMs)

Abstract State Machines: ASM- Specification's method

- Fundamentals
- Sequential algorithms
- ASM-Specifications

Distributed ASM: Concurrency, reactivity, time

- Fundamentals: Orders, CPO's, proof techniques
- Induction
- DASM
- Reactive and time-depending systems

Refinement

- Lecture Börger's ASM-Buch

Algebraic Specification

Algebraic Specification: Initial Semantics

Algebraic Specification: operationalization

Computability and Implementation

Role of formal Specifications

- Software and hardware systems must accomplish well defined tasks (**requirements**).
- Software Engineering has as goal
  - Definition of criteria for the evaluation of SW-Systems
  - Methods and techniques for the development of SW-Systems, that accomplish such criteria
  - Characterization of SW-Systems
  - Development processes for SW-Systems
  - Measures and Supporting Tools
- Simplified view of a SD-Process:
  Definition of a sequence of actions and descriptions for the SW-System to be developed. Process- and Product-Models

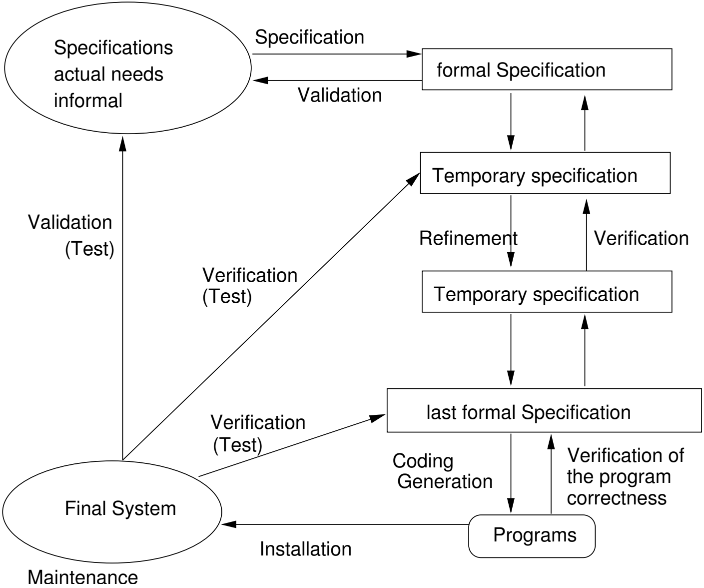
  Goal: The group of documents that includes an executable program.

Models for SW-Development

- Waterfall model, Spiral model,...

  Phases ≡ Activities + Product Parts (partial descriptions)
  In each stage of the DP

  Description: a SW specification, that is, a stipulation of what must be achieved, but not always how it is done.


Specifications
actual needs
informal
Specification
Validation
formal Specification
Temporary specification
Refinement
Verification
Temporary specification
Validation
(Test)
Verification
(Test)
Verification
(Test)
last formal Specification
Coding
Generation
Verification of
the program
correctness
Final System
Programs
Installation
Maintenance

Comment

- First Specification: Global Specification
  Fundament for the Development
  "Contract or Agreement" between Developers and Client
- Intermediate (partial) specifications:
  Base of the Communication between Developers.
- Programs: Final products.

Development paradigms

- Structured Programming
- Design + Program
- Transformation Methods
- ...

Properties of Specifications

Consistency Completeness

- Validation of the global specification regarding the requirements.
- Verification of intermediate specifications regarding the previous one.
- Verification of the programs regarding the specification.
- Verification of the integrated final system with respect to the global specification.
- Activities: Validation, Verification, Testing
  Consistency- and Completeness-Check
- Tool support needed!

Requirements

Functional	- -	non functional
what		time aspects
⋮		robustness
how		stability
		security
		adaptability
		ergonomics
		maintainability

Properties

Correctness: Does the implemented System fulfill the Requirements?

Test Validate Verify

Validation - Verification

From Wikipedia, the free encyclopedia
In common usage, **validation** is the process of checking if something satisfies a certain criterion. Examples would include checking if a statement is true (validity), if an appliance works as intended, if a computer system is secure, or if computer data are compliant with an open standard. Validation implies one is able to document that a solution or process is correct or is suited for its intended use.
In engineering or as part of a quality management system, **validation** confirms that the needs of an external customer or user of a product, service, or system are met. **Verification** is usually an internal quality process of determining compliance with a regulation, standard, or specification. An easy way of recalling the difference between validation and verification is that
validation is ensuring “you built the right product” and
verification is ensuring “you built the product right.”
Validation is testing to confirm that it satisfies user’s needs.

Requirements

- The global specification describes, as exact as possible, what must be done.
- **Abstraction of the *how***
  Advantages
  - apriori: Reference document, compact and legible.
  - aposteriori: Possibility to follow and document design decisions ⇝ **traceability, reusability, maintenance**.
- Problem: Size and complexity of the systems.

  Principles to be supported
  - Refinement principle: Abstraction levels
  - Structuring mechanisms
    Decomposition and modularization principles
  - Object orientation
  - Verification and validation concepts

Requirements Description $\rightsquigarrow$ Specification Language

- Choice of the specification technique depends on the System. Frequently more than a single specification technique is needed. (What – How).
- Type of Systems:
  Pure function oriented (I/O), reactive- embedded- real time- systems.
- Problem : Universal Specification Technique (UST)
  difficult to understand, ambiguities, tools, size ...
  e.g. UML
- Desired: Compact, legible and exact specifications

Here: formal specification techniques

Formal Specifications

- A specification in a formal specification language defines all the possible behaviors of the specified system.
- 3 Aspects: Syntax, Semantics, Inference System
  - Syntax: What's allowed to write: Text with structure, Properties often described by formulas from a logic.
  - Semantics: Which models are associated with the specification, $\rightsquigarrow$ specification models.
  - Inference System: Consequences (Derivation) of properties of the system. $\rightsquigarrow$ Notion of consequence.

Formal Specifications

- Two main classes:

Model oriented	-	-	Property oriented
(constructive)			(declarative)
e.g.VDM, Z, ASM			signature (functions, predicates)
Construction of a non-ambiguous model from available data structures and construction rules			Properties (formulas, axioms)
			models
			algebraic specification
Concept of correctness			AFFIRM, OBJ, ASF,...

- Operational specifications:
  Petri nets, process algebras, automata based (SDL).

Specifications: What for?

- The concept of program correctness is not well defined without a formal specification.
- A verification is not possible without a formal specification.
- Other concepts, like the concept of refinement, simulation become well defined.

Wish List

- Small gap between specification and program: Generators, Transformators.
- Not too many different formalisms/notations.
- Tool support.
- Rapid prototyping.
- Rules for "constructing" specifications, that guarantee certain properties (e.g. consistency + completeness).

Formal Specifications

- Advantages:
  - The concepts of correctness, equivalence, completeness, consistency, refinement, composition, etc. are treated in a mathematical way (based on the logic)
  - Tool support is possible and often available
  - The application and interconnection of different tools are possible.
- Disadvantages:

Refinements

Abstraction mechanisms

- Data abstraction (representation)
- Control abstraction (Sequence)
- Procedural abstraction (only I/O description)

Refinement mechanisms

- Choose a data representation (sets by lists)
- Choose a sequence of computation steps
- Develop algorithm (Sorting algorithm)

Concept: Correctness of the implementation

- Observable equivalences
- Behavioral equivalences

Structuring

Problems: Structuring mechanisms

- Horizontal:
  Decomposition/Aggregation/Combination/Extension/
  Parameterization/Instantiation
  (Components)

  Goal: Reduction of complexity, Completeness
- Vertical:
  Realization of Behavior
  Information Hiding/Refinement

  Goal: Efficiency and Correctness

Tool support

- Syntactic support (grammars, parser,...)
- Verification: theorem proving (proof obligations)
- Prototyping (executable specifications)
- Code generation (out of the specifications generate C code)
- Testing (from the specification generate test cases for the program)

Desired:

To generate the tools out of the syntax and semantics of the specification language

Example: declarative

Example 2.1. *Restricted logic: e.g. equational logic*

- *Axioms*: $\forall X \; t_1 = t_2 \qquad t_1, t_2$ *terms.*
- *Rules*: *Equals are replaced with equals. (directed).*
- *Terms* $\approx$ *names for objects (identifier), structuring, construction of the object.*
- *Abstraction*: *Terms as elements of an algebra, term algebra.*

Example: declarative

Foundations for the algebraic specification method:

- Axioms induce a congruence on a term algebra
- Independent subtasks
  - Description of properties with equality axioms
  - Representation of the terms
- Operationalization
  - spec, t term give out the „value“ of t, i.e. $t' \in \text{Value(spec)}$ with spec $\models t = t'$.
  - $\rightsquigarrow$ Functional programming: LISP, CAML,...
    $F(t_1, \ldots, t_n)$ eval( ) $\rightsquigarrow$ value.

Example: Model-based constructive: VDM

Unambiguous (Unique model), standard (notations),
Independent of the implementation, formally manipulable, abstract, structured, expressive, consistency by construction

Example 2.2. *Model (state)-based specification technique VDM*

- *Based on naive set theory, PL 1, preconditions and postconditions.*
  Primitive types: $\mathbb{B}$ *Boolean* $\{true, false\}$, $\mathbb{N}$ *natural* $\{0, 1, 2, 3, \ldots\}$, $\mathbb{Z}, \mathbb{R}$
- *Sets:* $\mathbb{B}$*-Set: Sets of* $\mathbb{B}$*-'s.*
- *Operations on sets:* $\in$*: Element, Element-Set* $\to \mathbb{B}$, $\cup, \cap, \setminus$
- *Sequences:* $\mathbb{Z}^*$*: Sequences of integer numbers.*
- *Sequence operations:* $\frown$*: Sequences, Sequences* $\to$ *Sequences.* *„Concatenation"*
  e.g. $[\ ] \frown [true, false, true] = [true, false, true]$
  len: sequences $\to \mathbb{N}$*,* *hd: sequences* $\rightsquigarrow$ *elem (partial).*
  tl: sequences $\rightsquigarrow$ *sequences,* *elem: sequences* $\to$ *Elem-Set.*

Operations in VDM

See e.g.: http://www.vdmportal.org/twiki/bin/view
VDM-SL: System State, Specification of operations

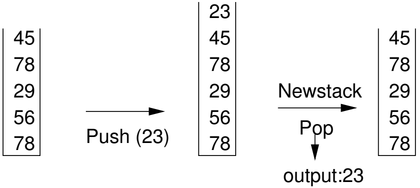
Format:

Operation-Identifier (Input parameters) Output parameters
Pre-Condition
Post-Condition

e.g.
$Int\_SQR(x : \mathbb{N})z : \mathbb{N}$
pre $x \geq 1$
post $(z^2 \leq x) \wedge (x < (z+1)^2)$

Example VDM: Bounded stack

Example 2.3.
- *Operations:* · *Init* · *Push* · *Pop* · *Empty* · *Full*



Contents $= \mathbb{N}^*$ Max_Stack_Size $= \mathbb{N}$

- STATE STACK OF
  s : Contents
  n : Max_Stack_Size
  inv : mk-STACK$(s, n) \triangleq$ len $s \leq n$
  END

Bounded stack

$\text{Init}(\text{size} : \mathbb{N})$
ext wr s : Contents
wr n : Max_Stack_Size
pre true
post $s = [\ ] \land n = \text{size}$

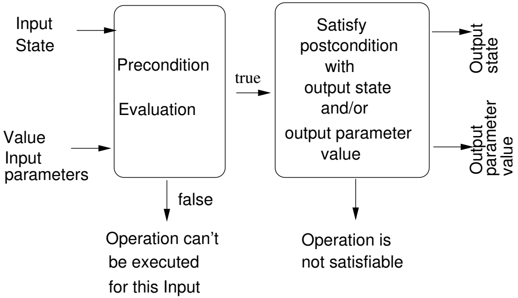
$\text{Full}(\ )b : \mathbb{B}$
ext rd s : Contents
rd n : Max_Stack_Size
pre true
post $b \Leftrightarrow (\text{len}\, s = n)$

$\text{Push}(c : \mathbb{N})$
ext wr s : Contens
rd n : Max_Stack_Size
pre $\text{len}\, s < n$
post $s = [c] \frown \overleftarrow{s}$

$\text{Pop}(\ )c : \mathbb{N}$
ext wr s : Contens
pre $\text{len}\, s > 0$
post $\overleftarrow{s} = [c] \frown s$

$\rightsquigarrow$ Proof-Obligations

General format for VDM-operations



General form VDM-operations

Proof obligations:
For each acceptable input there's (at least) one acceptable output.

$$\forall s_i, i \cdot (\text{pre-op}(i, s_i) \Rightarrow \exists s_o, o \cdot \text{post-op}(i, s_i, o, s_o))$$

When there are state-invariants at hand:

$$\forall s_i, i \cdot (\text{inv}(s_i) \land \text{pre-op}(i, s_i) \Rightarrow \exists s_o, o \cdot (\text{inv}(s_o) \land \text{post-op}(i, s_i, o, s_o)))$$

alternatively

$$\forall s_i, i, s_o, o \cdot (\text{inv}(s_i) \land \text{pre-op}(i, s_i) \land \text{post-op}(i, s_i, o, s_o) \Rightarrow \text{inv}(s_o))$$

See e.g. Turner, McCluskey The Construction of Formal Specifications or Jones C.B. Systematic SW Development using VDM Prentice Hall.

Stack: algebraic specification

Example 2.4. *Elements of an algebraic specification: Signature (sorts, operation names with the arity), Axioms (often only equations)*

SPEC *STACK*
USING *NATURAL, BOOLEAN* *"Names of known SPECs"*
SORT *stack* *"Principal type"*
OPS *init* : $\rightarrow$ *stack* *"Constant of the type stack, empty stack"*
push : *stack nat* $\rightarrow$ *stack*
pop : *stack* $\rightarrow$ *stack*
top : *stack* $\rightarrow$ *nat*
is_empty? : *stack* $\rightarrow$ *bool*
stack_error : $\rightarrow$ *stack*
nat_error : $\rightarrow$ *nat*

(Signature fixed)

Axioms for Stack

FORALL s : stack n : nat
AXIOMS

is_empty? (init) = true
is_empty? (push (s, n)) = false
pop (init) = stack_error
pop (push (s, n)) = s
top (init) = nat_error
top (push (s,n)) = n

Terms or expressions:
top (push (push (init, 2), 3)) "means" 3
How is the "bounded stack" specified algebraically?
Semantics? Operationalization?

Variant: Z and B- Methods: Specification-Development-Programs.

- Covering: Technical specification (what), development through refinement, architecture (layers' architecture), generation of executable code.
- Proofs: Program construction $\equiv$ Proof construction. Abstraction, instantiation, decomposition.
- Abstract machines: Encapsulation of information (Modules, Classes, ADT).
- Data and operations: SWS is composed of abstract machines. Abstract machines „get " data and „offer" operations. Data can only be accessed through operations.

Z- and B- Methods: Specification-Development-Programs.

- Data specification: Sets, relations, functions, sequences, trees. Rules (static) with help of invariants.
- Operator specification: not executable „pseudocode“.
  Without loops:
  Precondition + atomic action
  PL1 generalized substitution
- Refinement ($\leadsto$ implementation).
- Refinement (as specification technique).
- Refinement techniques:
  Elimination of not executable parts, introduction of control structures (cycles).
  Transformation of abstract mathematical structures.

Z- and B- Methods: Specification-Development-Programs.

- Refinement steps: Refinement is done in several steps.
  Abstract machines are newly constructed. Operations for users remain the same, only internal changes.
  In-between steps: Mix code.
- Nested architecture:
  Rule: not too many refinement steps, better apply decomposition.
- Library: Predefined abstract machines, encapsulation of classical DS.
- Reusability
- Code generation: Last abstract machine can be easily translated into a program in an imperative Language.

Z- and B- Methods: Specification-Development-Programs.

Important here:

- Notation: Theory of sets + PL1, standard set operations, Cartesian product, power sets, set restrictions $\{x \mid x \in s \wedge P\}$, P predicate.
- Schemata (Schemes) in Z Models for declaration and constraint {state descriptions}.
- Types.
- Natural Language: Connection Math objects $\rightarrow$ objects of the modeled world.
- See Abrial: The B-Book,
  Potter, Sinclair, Till: An Introduction to Formal Specification and Z,
  Woodcock, Davis: Using Z Specification, Refinement, and Proof $\rightsquigarrow$ Literature