«UML» 2002

HOL-OCL: Experiences, Consequences and Design Choices

Achim D. Brucker and Burkhart Wolff
Albert-Ludwigs Universität Freiburg, Germany

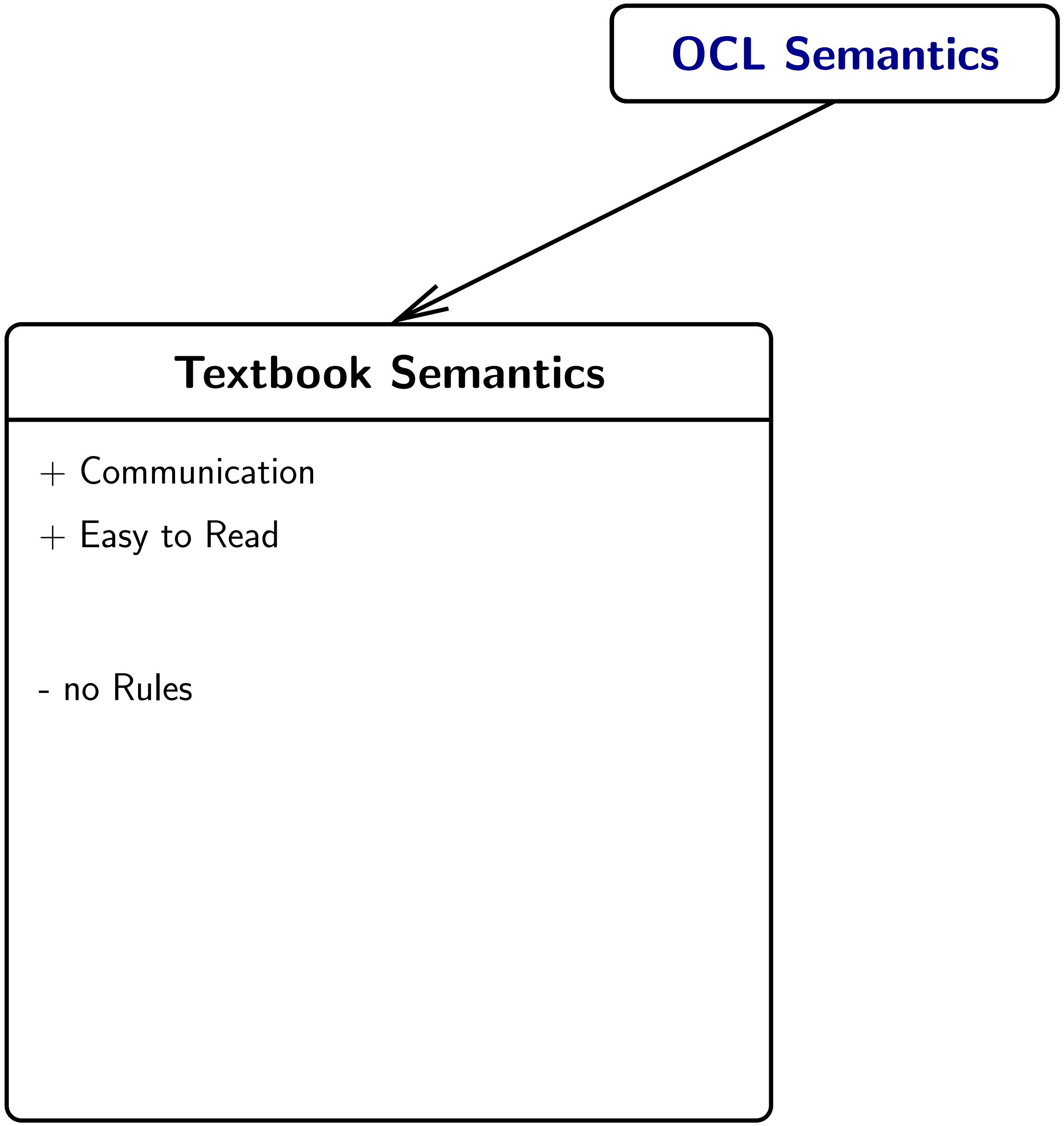
October 3, 2002

This work was partially funded by the OMG member Interactive Objects Software GmbH.

Roadmap

1. **Motivation:** Use of Semantics
2. **Foundations:** Isabelle/HOL, *HOL-OCL*
3. ***HOL-OCL*:** Experiences and Applications
4. **Conclusion**

OCL Semantics


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules

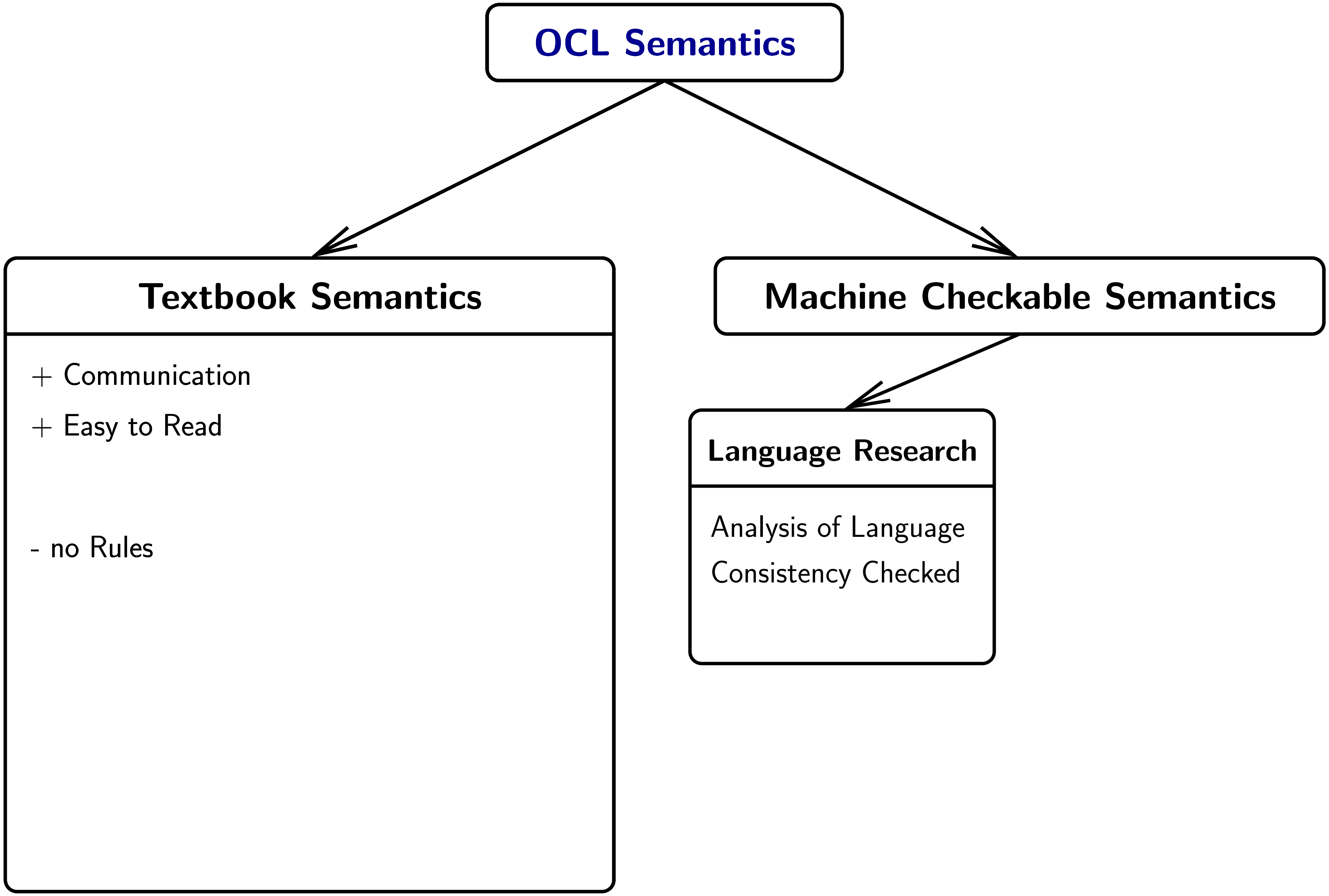
OCL Semantics

Textbook Semantics

+ Communication
+ Easy to Read

- no Rules

Machine Checkable Semantics


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked

OCL Semantics

Textbook Semantics

+ Communication
+ Easy to Read

- no Rules

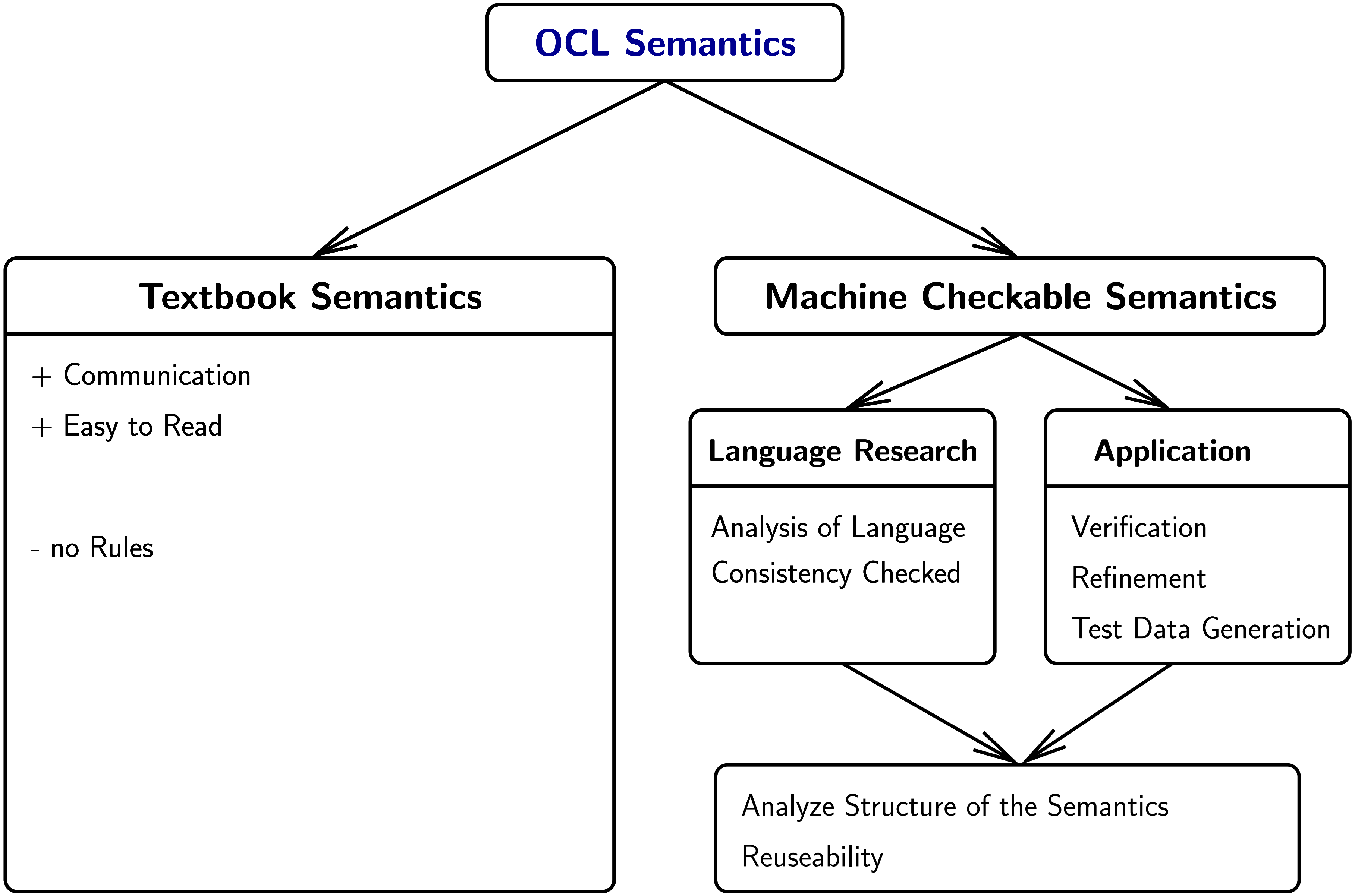
Machine Checkable Semantics

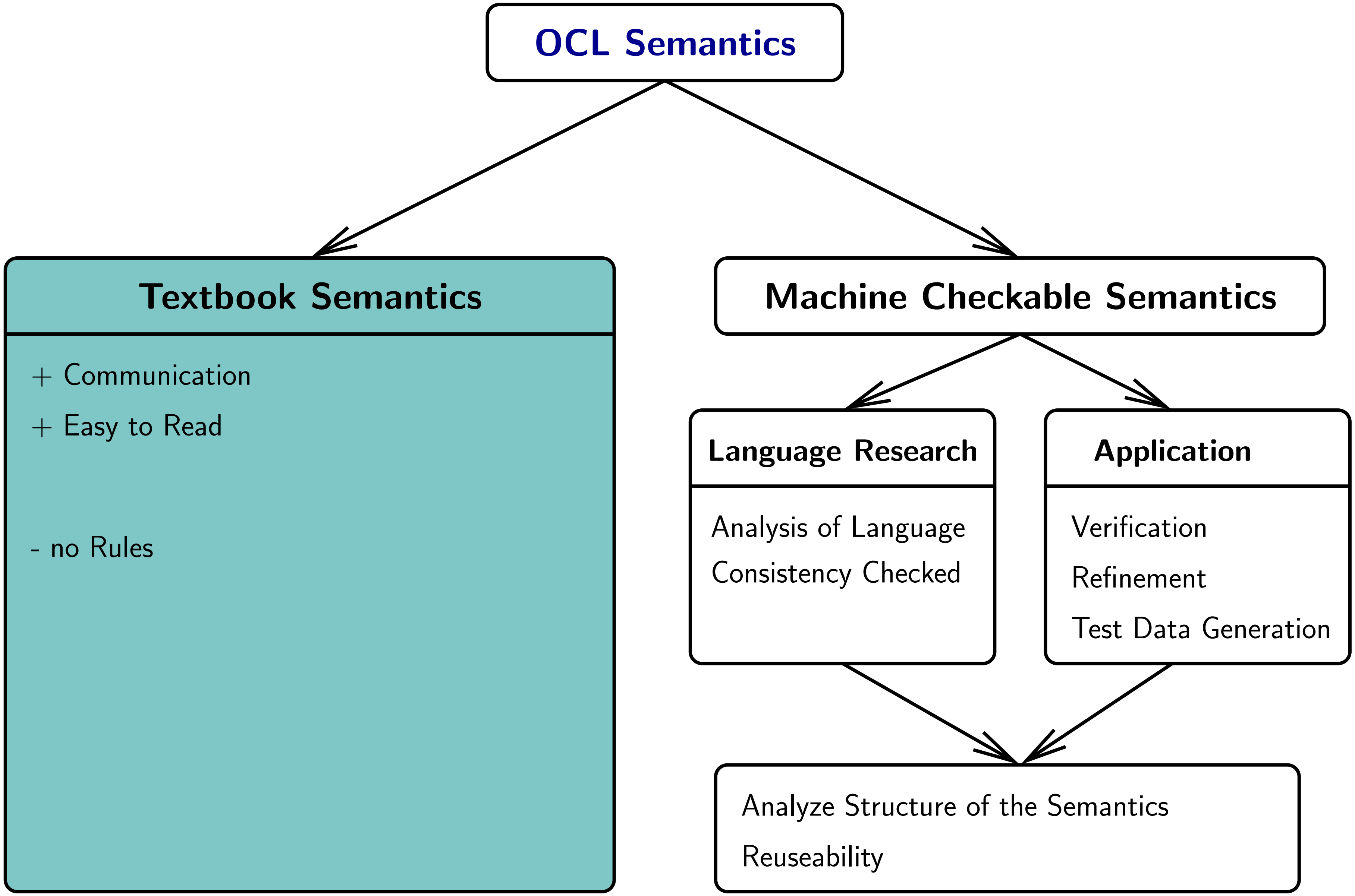
Language Research

Analysis of Language
Consistency Checked

Application

Verification
Refinement
Test Data Generation


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability

Textbook Semantics: An Example

- The interpretation of the logical and is given by a truth-table:

a	b	a and b
false	false	false
false	true	false
false	$\bot_{\mathscr{L}}$	false

a	b	a and b
true	false	false
true	true	true
true	$\bot_{\mathscr{L}}$	$\bot_{\mathscr{L}}$

a	b	a and b
$\bot_{\mathscr{L}}$	false	false
$\bot_{\mathscr{L}}$	true	$\bot_{\mathscr{L}}$
$\bot_{\mathscr{L}}$	$\bot_{\mathscr{L}}$	$\bot_{\mathscr{L}}$

- The Interpretation of "X->union(Y)" for sets ("$X \cup Y$"):

$$I(\cup)(X,Y) \equiv \begin{cases} X \cup Y & \text{if } X \neq \bot_{\mathscr{L}} \text{ and } Y \neq \bot_{\mathscr{L}} \\ \bot_{\mathscr{L}} & \text{otherwise} \end{cases}$$

This is a **strict** and **lifted** version of the union of "mathematical sets".

Textbook Semantics

- "Paper-and-Pencil" work in mathematical notation.

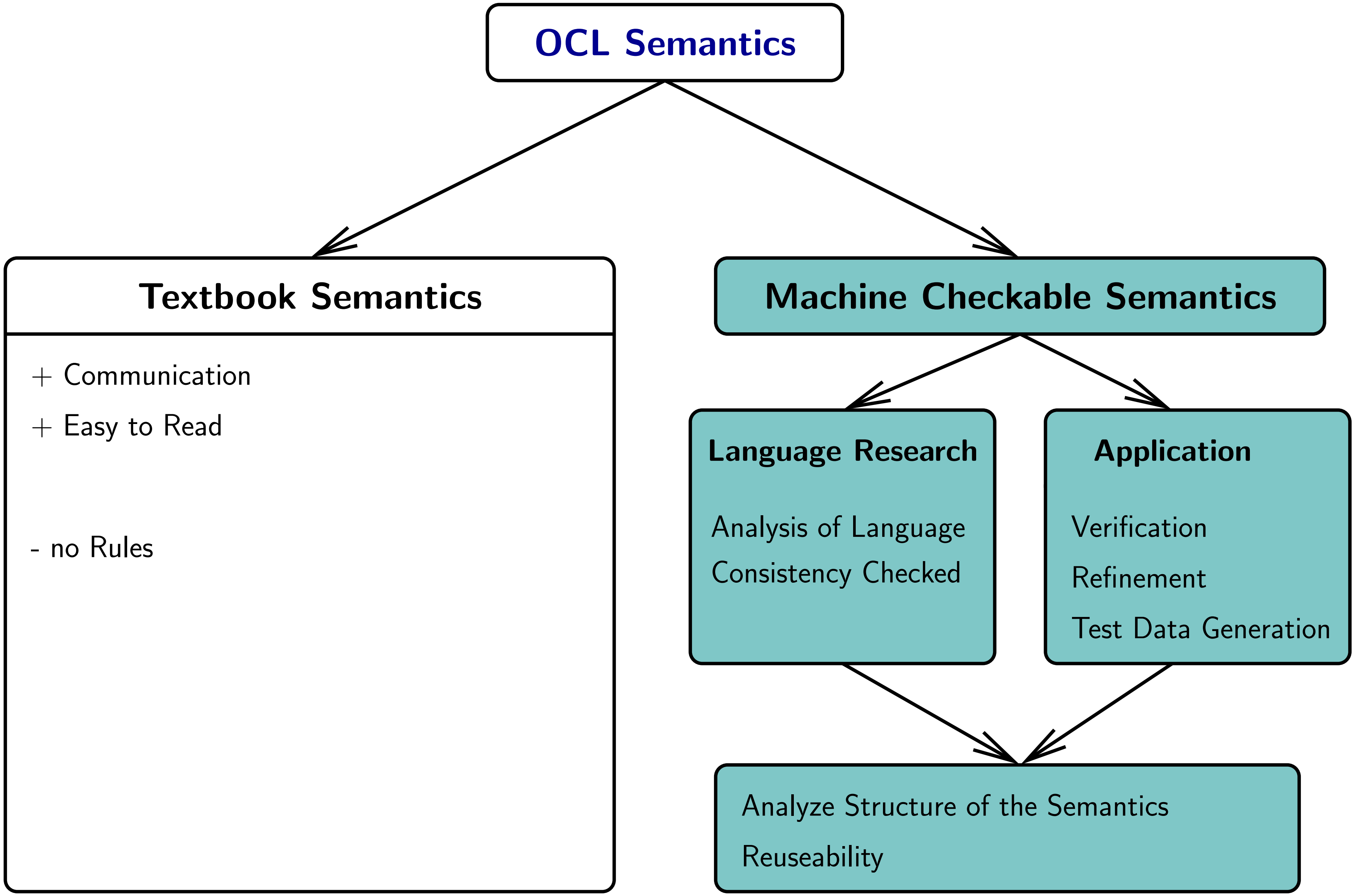
(+) Useful to communicate semantics.

(+) Easy to read.

(−) No rules, no laws.

(−) Informal or meta-logic definitions (*"The Set is the mathematical set."*).

(−) It is easy to write inconsistent semantic definitions.


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability

Machine-Checkable Semantics

Motiviation: Honor the semantical structure of the language.

- A machine-checked semantics
  - conservative embeddings guarantee **consistency** of the semantics.
  - builds the basis for **analyzing** language features.
  - allows incremental changes of semantics.
- As basis of further tool support for
  - **reasoning** over specifications.
  - **refinement** of specifications.
  - automatic **test data generation**.

Machine Checkable Semantics

- The definition of the logical *and* (Kleene-logic):

$$\begin{aligned} S \ \textbf{and}\ T \equiv \lambda c.\ &\textbf{if}\ DEF\ (S\ c)\ \textbf{then} \\ &\qquad \textbf{if}\ DEF\ (T\ c)\ \textbf{then}\ \lfloor \lceil S\ c \rceil \wedge \lceil T\ c \rceil \rfloor \\ &\qquad \textbf{else}\ \textbf{if}\ S\ c = (\lfloor False \rfloor)\ \textbf{then}\ \lfloor False \rfloor\ \textbf{else}\ \bot \\ &\textbf{else}\ \textbf{if}\ T\ c = (\lfloor False \rfloor)\ \textbf{then}\ \lfloor False \rfloor\ \textbf{else}\ \bot \end{aligned}$$

  The truth-table can be derived from this definition.

- The *union* of sets is defined as the **strict** and **lifted** version of $\cup$:

$$\begin{aligned} \text{union} \equiv &\text{lift}_2(\text{strictify}_N(\lambda X.\ \text{strictify}_N(\\ &\qquad \lambda Y.\ \text{Abs\_SSet}\ (\lfloor \lceil \text{Rep\_SSet}\ X \rceil \cup \lambda \lceil \text{Rep\_SSet}\ Y \rceil \rfloor)))) \end{aligned}$$

- These definitions can be automatically rewritten into "Textbook-style".

Foundations: Using Isabelle/HOL for defining semantics

- Foundation:
  - **Isabelle** is a generic theorem prover.
  - **Higher-order logic (HOL)** is a classical logic with higher-order functions.
  - **Isabelle**'s logics: designed for extensible.
- Defining semantics via extending logics can be done
  - by a **deep embedding** or a **shallow embedding**.

    Shallow: Direct definition of the semantics, e.g. each construct is represented by some function on a semantic domain.

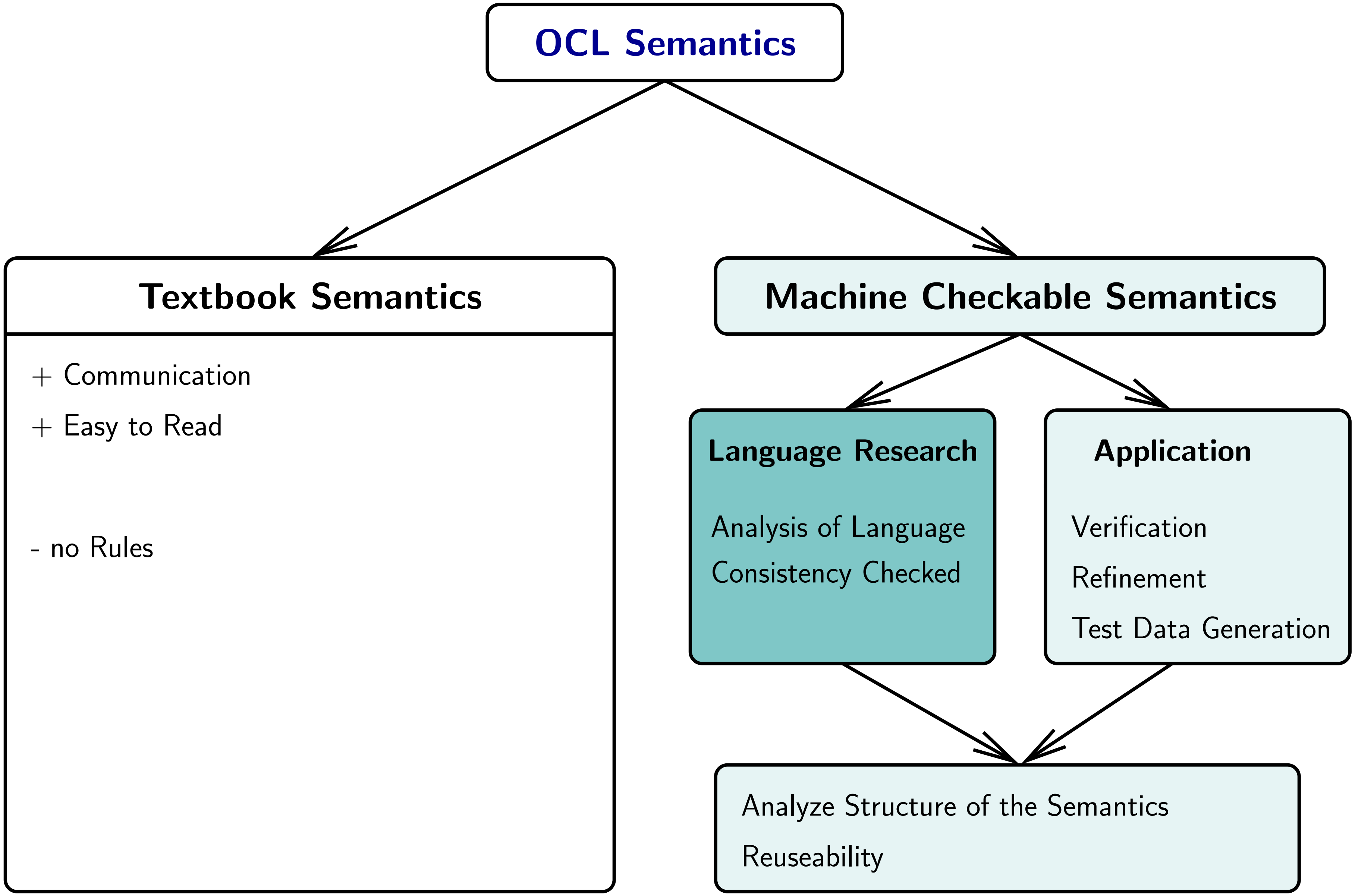
    Deep: The abstract syntax is presented as a datatype and a semantic function I from syntax to semantics.
  - by introducing **new axioms** or by **conservative** (proving new properties) extensions.

HOL-OCL: A Shallow Embedding of OCL into HOL

- is build on top of Isabelle/HOL.
- is a shallow embedding of OCL into HOL.
- provides a consistent (machine checked) OCL semantics.
- allows the examination of OCL features.
- builds the basis for OCL tool development.
- follows OCL 1.4 and the RfP for OCL 2.0
- over 2000 theorems (language properties) proven.

The Technical Design of *HOL-OCL*

- **Reuseability:**
  - Reuse old proofs for class diagrams constructed via inheritance introduction of new classes.
  - Extendible semantics approach.
- **Representing semantics structurally:**
  - Organize semantic definitions by certain combinators capturing the semantical essence (e.g. lifting and strictness).
  - Automatically construct theorems out of uniform definitions.


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability

HOL-OCL Language Research: Smashed Sets

For handling undefined elements ($\perp_{\mathscr{L}}$) in Sets we have two possibilities:

1. Not smashed:

$$\{X, \perp_{\mathscr{L}}\} \neq \perp_{\mathscr{L}} \text{ with the consequence } X \in \{X, \perp_{\mathscr{L}}\} \text{ and } \perp_{\mathscr{L}} \in \{X, \perp_{\mathscr{L}}\}$$

2. Smashed:

$$\{X, \perp_{\mathscr{L}}\} = \perp_{\mathscr{L}} \text{ with the consequence } X \notin \{X, \perp_{\mathscr{L}}\} \text{ and } \perp_{\mathscr{L}} \notin \{X, \perp_{\mathscr{L}}\}$$

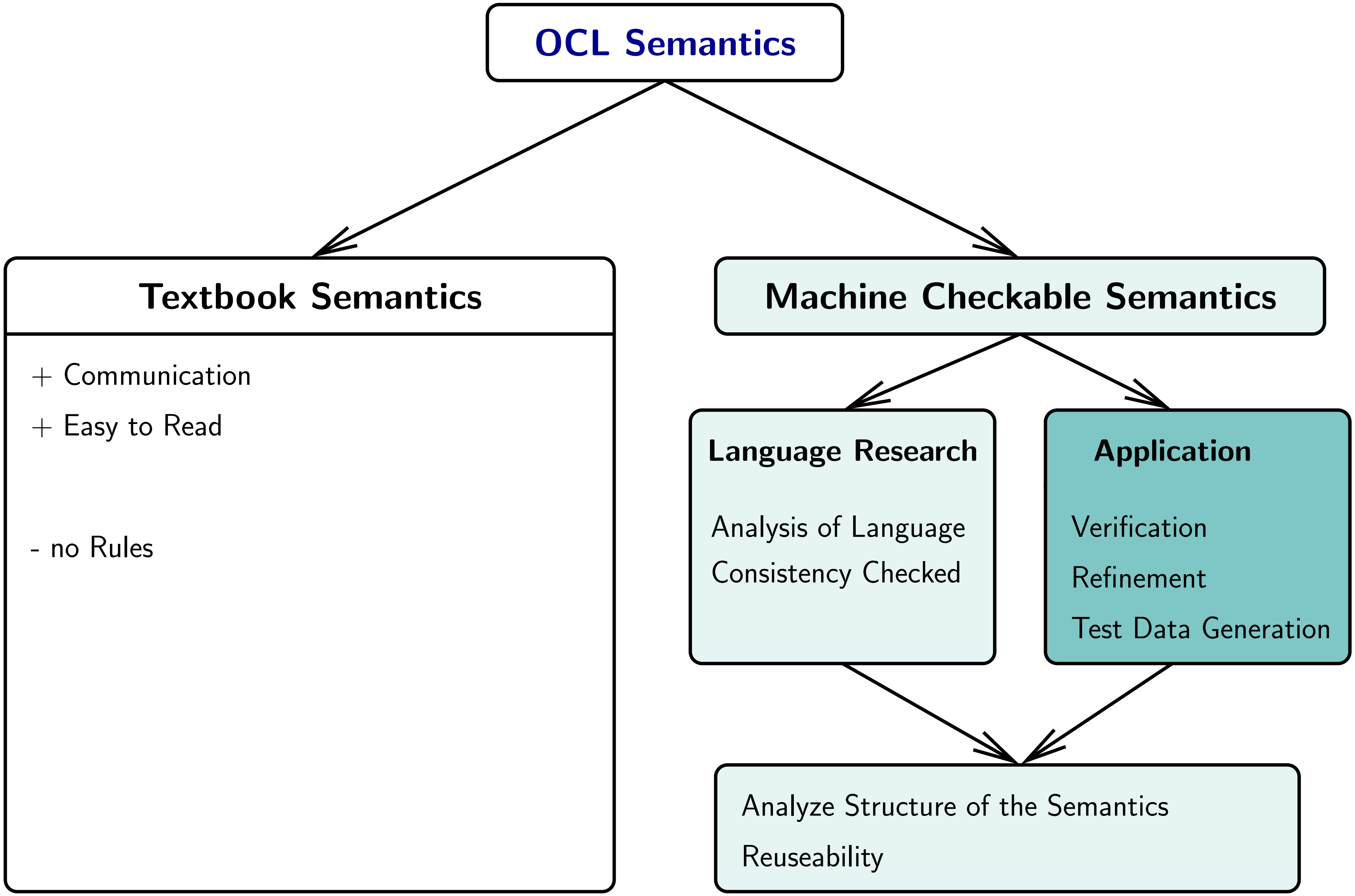
HOL-OCL Language Research: Smashed Sets

- The OCL 2.0 proposal suggest **not smashed** Sets, Bags, Sequences and Tuples:

$$I(count : Set(t) \times tInteger)(s, v) = \begin{cases} 1 & \text{if } v \in s \\ 0 & \text{if } v \notin s \\ \bot_{\mathscr{L}} & \text{if } s = \bot_{\mathscr{L}} \end{cases}$$

  And therefore “X->includes(Y)” is **not executable**!

- We encourage the use of **smashed** Sets, Bags, Sequences and Tuples:
  - This mirrors the operational behavior of programming languages (e.g. Java).
  - This allows the definition of a executable OCL subset.


OCL Semantics
Textbook Semantics
+ Communication
+ Easy to Read
- no Rules
Machine Checkable Semantics
Language Research
Analysis of Language
Consistency Checked
Application
Verification
Refinement
Test Data Generation
Analyze Structure of the Semantics
Reuseability

HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.

Triangle

+ isTriangle(s0, s1, s2: Integer): Boolean
+ triangle(s0, s1, s2: Integer): TriType

<<Enumeration>>
TriangType

invalid
scalene
isosceles
equilateral

```
context
 Triangle::isTriangle(s0,s1,s2:Integer):Boolean

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result =       (s2 < (s0 + s1))
            and (s0 < (s1 + s2))
            and (s1 < (s0 + s2))
```

HOL-OCL Application: Test Data Generation

Based on a UML/OCL specification a minimal set of test data is calculated which can be used for validating an implementation.

Triangle
+ isTriangle(s0, s1, s2: Integer): Boolean + triangle(s0, s1, s2: Integer): TriType

<<Enumeration>> **TriangType**
invalid scalene isosceles equilateral

```
context
 Triangle::triangle(s0,s1,s2:Integer):TriangType

pre:
 (s0 > 0) and (s1 > 0) and (s2 > 0)

post:
 result = if (isTriangle(s0,s1,s2)) then
            if (s0 = s1) then
               if (s1 = s2) then
                 Equilateral::TriangType
               else
                 Isosceles::TriangType endif
            else
              if (s1 = s2) then
                Isosceles::TriangType
              else
                if (s0 = s2) then
                  Isosceles::TriangType
                else
                  Scalene::TriangType
             endif endif endif
          else
            Invalid::TriangType endif
```

HOL-OCL Application: Test Data Generation

1. Reduce all logical operation to the basis operators:

   and, **or**, und **not**

2. Determine disjunctive normal Form (DNF):

   $$x \text{ and } (y \text{ or } z) \rightsquigarrow (x \text{ and } y) \text{ or } (x \text{ and } z)$$

3. Eliminate unsatisfiable sub-formulae, e.g.:

   scalene and invalid

4. Select test data with respect to boundary cases.

Partitioning of the Test Data

$\text{triangle}\ s_0\ s_1\ s_2 = @result\bullet \vDash$

$result \triangleq \text{invalid and not isTriangle}\ s_0\ s_1\ s_2$

or

$result \triangleq \text{equilateral and isTriangle}\ s_0\ s_1\ s_2 \text{ and } s_0 \triangleq s_1 \text{ and } s_1 \triangleq s_2$

or

$result \triangleq \text{isosceles and isTriangle}\ s_0\ s_1\ s_2 \text{ and } s_0 \triangleq s_1 \text{ and } s_1 \not\triangleq s_2$

or

$result \triangleq \text{isosceles and isTriangle}\ s_0\ s_1\ s_2 \text{ and } s_0 \triangleq s_2 \text{ and } s_0 \not\triangleq s_1$

or

$result \triangleq \text{isosceles and isTriangle}\ s_0\ s_1\ s_2 \text{ and } s_1 \triangleq s_2 \text{ and } s_0 \not\triangleq s_1$

or

$result \triangleq \text{scalene and isTriangle}\ s_0\ s_1\ s_2 \text{ and } s_0 \not\triangleq s_1 \text{ and } s_0 \not\triangleq s_2 \text{ and } s_1 \not\triangleq s_2$

Partitioning of the Test Data

1. Input describes **no** triangle.

2. Input describes an **equilateral** triangle.

3. Input describes an **isosceles** triangle:

   (a) with s_0 equals s_1.

   (b) with s_0 equals s_2.

   (c) with s_1 equals s_2.

4. Input describes an **scalene** triangle.

For each partition, concrete test data has to be selected with respect to boundary cases (e.g. max./min. Integers, . . .).

Conclusion

A theorem prover based OCL definition of the OCL semantics:

- provides a sound and consistent semantic "Textbook".
- allows the definition of a proof calculi over OCL.
- Gives OCL/UML the power of well-known Formal Methods (e.g. Z, VDM), e.g. for:
  - validation..
  - verification.
  - Refinement.
  - automated test data generation.
  - ...

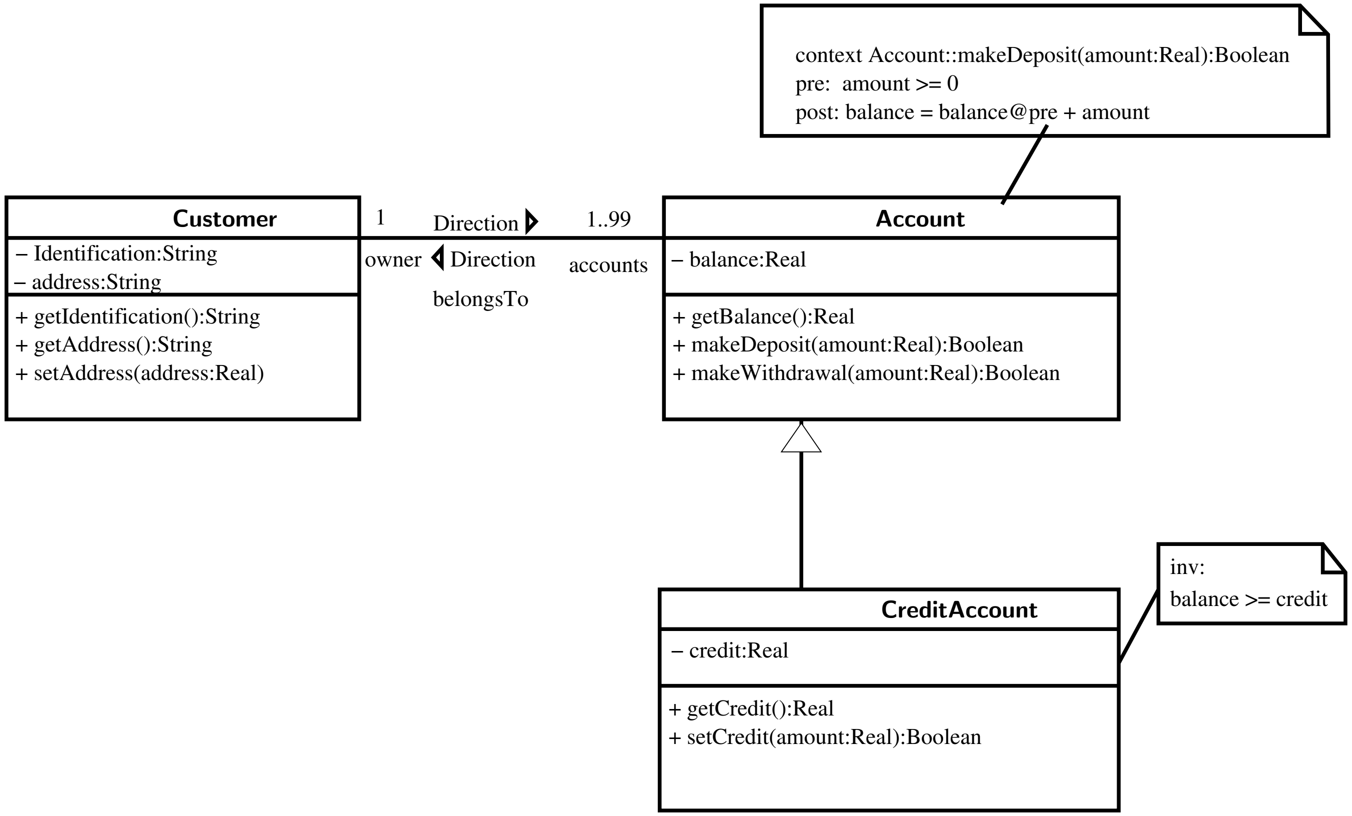
Conclusion: Tabular overview

	OCL 1.4	OCL 2.0 RfP	*HOL-OCL* preference
extendible universes	☐	☐	☑
general recursion	☐	☐	☑
smashing	?	☐	☑
automated flattening	☑	☐	☐
tuples	☐	☑	☑
finite state	☑	☑	☐
general Quantifiers	☐	☐	☑
allInstances finite	☑	☑	☐
Kleene logic	☑	☑	☑
strong and weak equality	☐	☑	☑

Appendix

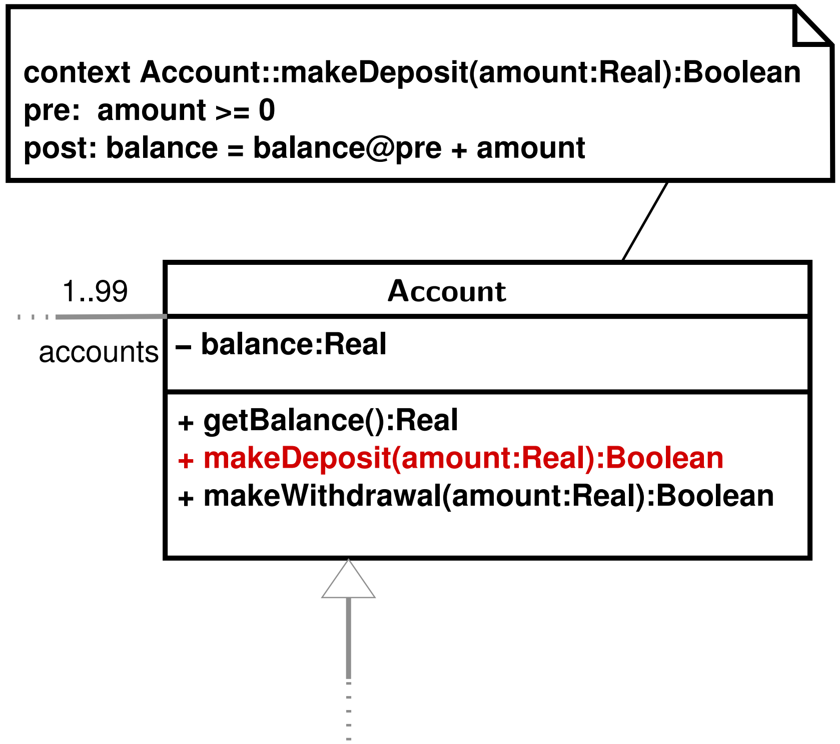
The Unified Modeling Language (UML)

- diagrammatic OO modeling language
- many diagram types, e.g.
  - class diagrams (static)
  - state charts (dynamic)
  - use cases
- semantics currently standardized by the OMG
- we expect wide use in SE-Tools (ArgoUML, Rational Rose,...)



The Object Constraint Language (OCL)

- designed for annotating UML diagrams (and give foundation for injectivities, . . .)
- based on logic and set theory
- in the context of class–diagrams:
  - preconditions
  - postconditions
  - invariants
- will be used for other diagram types too



Recursive Methods

OCL allows **recursive method** invocation "as long as the recursion is not infinite".

For handling non-terminating recursion two possibilities are possible:

- **It is forbidden:**
  - non-termination is undecidable
  - needs a notion of well-formedness
  - not machine-checkable
  - alternative: well-founded recursion (requires new syntactic and semantic concepts)
- **It is undefined ($\perp_{\mathscr{L}}$):**
  - consistent with **least-fixpoint** in the cpo-theory

Recursive Methods

- We encourage the use of recursive methods, because
  - they are executable
  - increase the expressive power of OCL
- But recursion comes not for free:
  - the semantics of method invocations needs to be clarified.
  - more complexity for code generation tools.

Invariants in OCL

Object Constraint Language Specification [?] (version 1.4), page 6-52

> An OCL expression is an invariant of the type and must be true for all instances of that type at any time.

- No problem, as we understand **at any time** as **at any reachable state.**
- Intermediate states violating this conditions have to be solved in the refinement notion.
- This also works with general recursion based on fix-points for query-functions.

On Executability of OCL

- The view of OCL as an object-oriented assertion language led to several restrictions, e.g.
  - allInstances() of basic data types is defined as $\perp_{\mathscr{L}}$.
  - states must be finite.
- Thus OCL is not self-contained.
- These restrictions hinder the definitions of general mathematical functions and theorems.
- We suggest to
  1. omit all these restrictions.
  2. define a **executable** OCL subset.

Shallow vs. Deep Embeddings

Representing the logical operations *or* and *and* via a

- **shallow embedding:**
  Direct definition of the semantics, e.g. each construct is represented by some function on a semantic domain.

- **deep embedding:**
  The abstract syntax is presented as a datatype and a semantic function I from syntax to semantics.

Shallow vs. Deep Embeddings

Representing the logical operations *or* and *and* via a

- **shallow embedding:**

$$x \,\mathsf{and}\, y \equiv \lambda\, e\,.\, x\ e \wedge y\ e \qquad x \,\mathsf{or}\, y \equiv \lambda\, e\,.\, x\ e \vee y\ e$$

- **deep embedding:**
  The abstract syntax is presented as a datatype and a semantic function I from syntax to semantics.

Shallow vs. Deep Embeddings

Representing the logical operations *or* and *and* via a

- **shallow embedding:**

$$x \,\mathsf{and}\, y \equiv \lambda\, e \,.\, x\ e \wedge y\ e \qquad x \,\mathsf{or}\, y \equiv \lambda\, e \,.\, x\ e \vee y\ e$$

- **deep embedding:**

$$expr = \mathsf{var}\ var \mid expr\ \mathsf{and}\ expr \mid expr\ \mathsf{or}\ expr$$

and the explicit semantic function I:

$$\begin{array}{rcl} I[\![\mathsf{var}\ x]\!] & = & \lambda\, e \,.\, e(x) \\ I[\![x\ \mathsf{and}\ y]\!] & = & \lambda\, e \,.\, I[\![x]\!]\ e \wedge I[\![y]\!]\ e \\ I[\![x\ \mathsf{or}\ y]\!] & = & \lambda\, e \,.\, I[\![x]\!]\ e \vee I[\![y]\!]\ e \end{array}$$

Contents