An invitation to the project
Denotational Engineering of Programming Languages

Andrzej Jacek Blikle

June 4th, 2021

Current working version of the book
"**A Denotational Engineering of Programming Languages**"
and current versions of transparences are available on
http://www.moznainaczej.com.pl/denotational-engineering/denotational-engineering-eng.
You can write to me on: andrzej.blikle@moznainaczej.com.pl

General rules of cooperation

Do not hesitate to:

- ask questions,
- doubt,
- question solutions and suggest you own,
- become an active coauthor of the project.

The philosophy of the project

What shall we try to do?

To suggest a way of improving the quality of programs.

THE QUALITY OF A PROGRAM:

1. the compliance of program-specification with user's expectations
2. the compliance of a program with its specifications.

Currently for Python-like sequential programming (no concurrency).

Work on concurrency is in progress

Why do we want to tackle the problem?

The state of the art in IT industry

An example of a disclamer *The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.*

The state of the art. in IT science

The KeY Book; From Theory to Practice (Springer 2016)
*For a long time, the term formal verification was almost synonymous with functional verification. In the last years, it became more and more clear that **full functional verification is an elusive goal for almost all application scenarios.** (…) Not verification but specification is the real bottleneck in functional verification.*

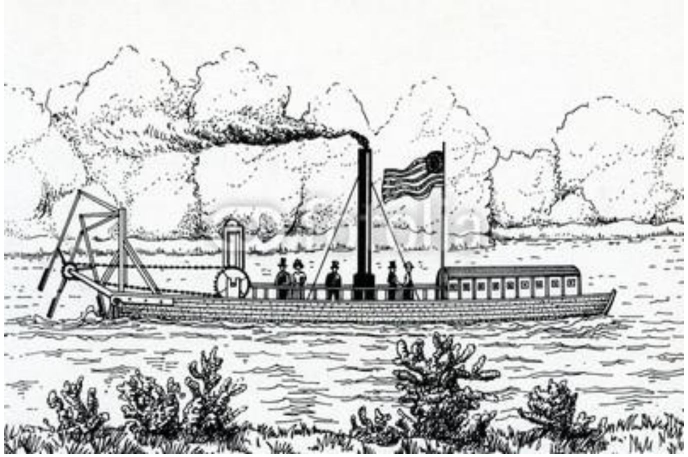
Why earlier attempts failed?

(although some experiments are still in the course)

In order to build a logic of programs,
we need a mathematical semantics of a programming language.

Two historical attempts to the definitions of mathematical semantics:

An operational semantics (VDL);
describe a virtual computer.



Denotational semantics (VDM)

$S : Language \longmapsto Denotations$

$S(P \blacklozenge Q) = S(P) \bullet S(Q)$

Ada and Chill, 1980
A subset of Pascal 1987

$S : AlgSyn \longmapsto AlgDen$

SEMANTICS
A homomorphism
between many-sorter algebras

Can a denotational semantics be written for any language?

My hypothesis
Probably not – at least not for the grammars of the languages that I know.

And certainly this hasn't been done so far.

A traditional approach to building denotational semantics:

First syntax:
how to talk about

→

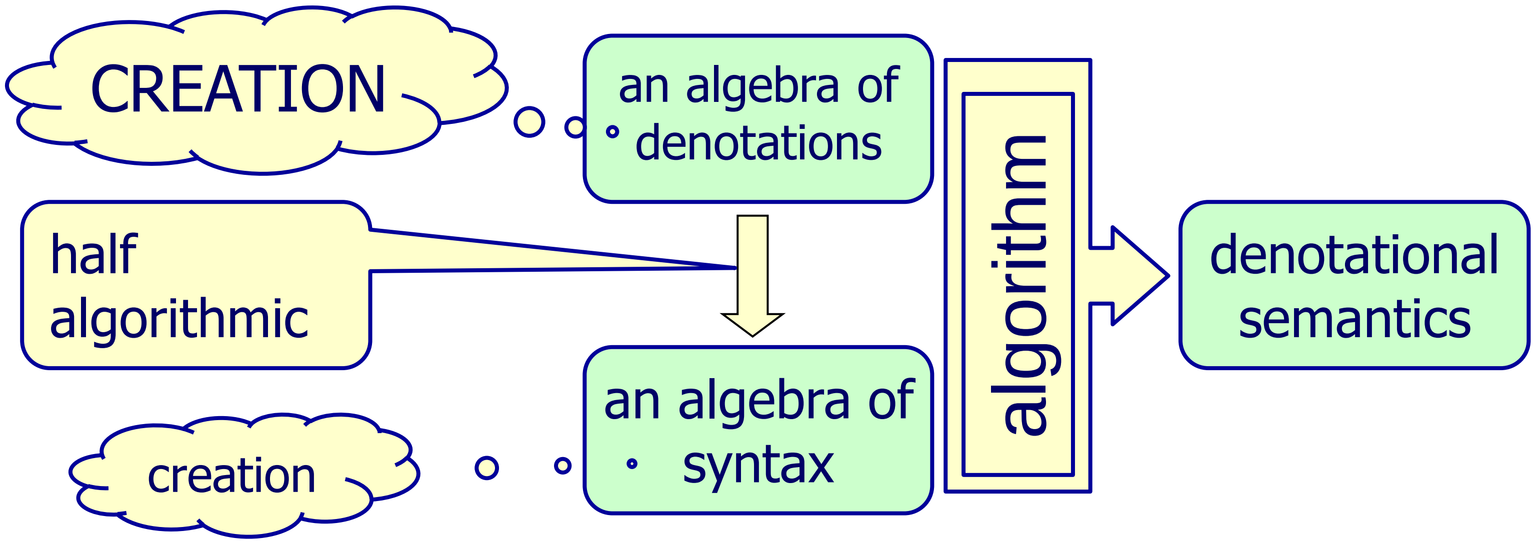
Then denotations
what to talk about

This order has a historical justification. When people started to think about semantics, syntaxes were already there.

Let's reverse the usual order of things

First describe the world of denotation: an algebra of the denotations of programs components.

Then derive from it an adequate corresponding syntax



While we have a languages with denotational semantics, we can think about proving programs correct.

Is proving programs correct a right way to validate programs?

Two problems:

1. A proof is usually longer then a theorem.
2. Programs are usually incorrect.

Let's reverse the usual order again

A mathematician
First a theorem, then a proof

An engineer
First a project (proof), then a product (e.g. a bridge)

Proof rules should be replaced by
sound program-construction rules

Validating programming

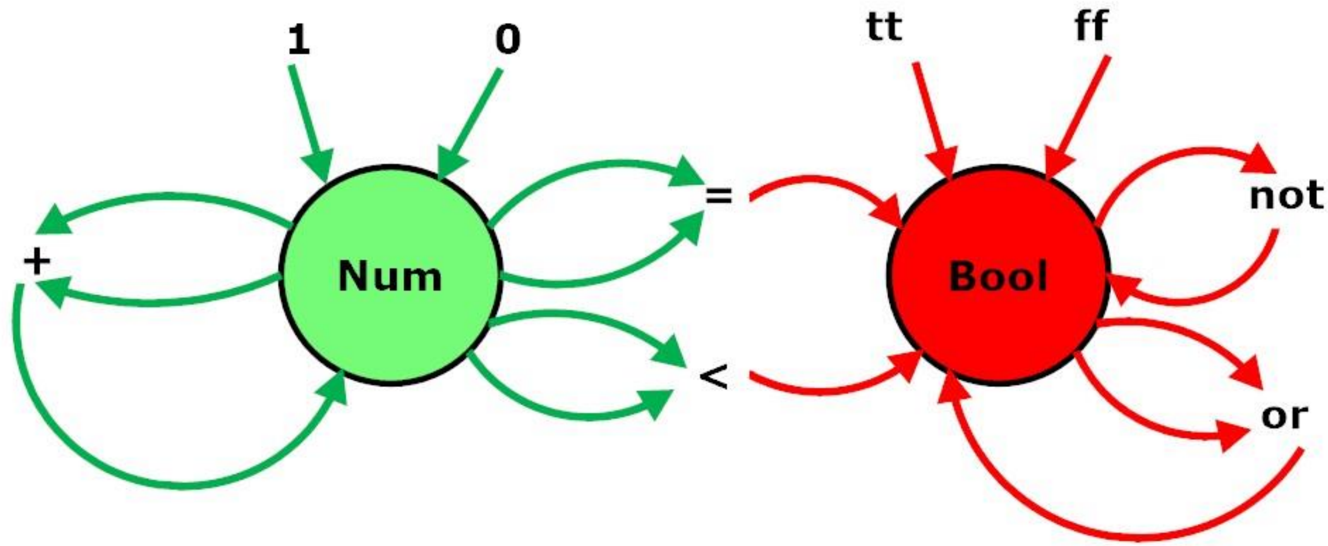
The general idea of a denotational model

These ideas have been published in my papers
in the years 1971 – 1989
(some with Antoni Mazurkiewicz and Andrzej Tarlecki)

MATHEMATICAL TOOLS

- fixed-point theory in CPO’s,
- set-theoretic domain equations (no Scott’s reflexive domains),
- three-valued predicate calculus,
- many-sorter algebras,
- abstract errors for error-handling mechanism.

An example of a many-sorted algebra



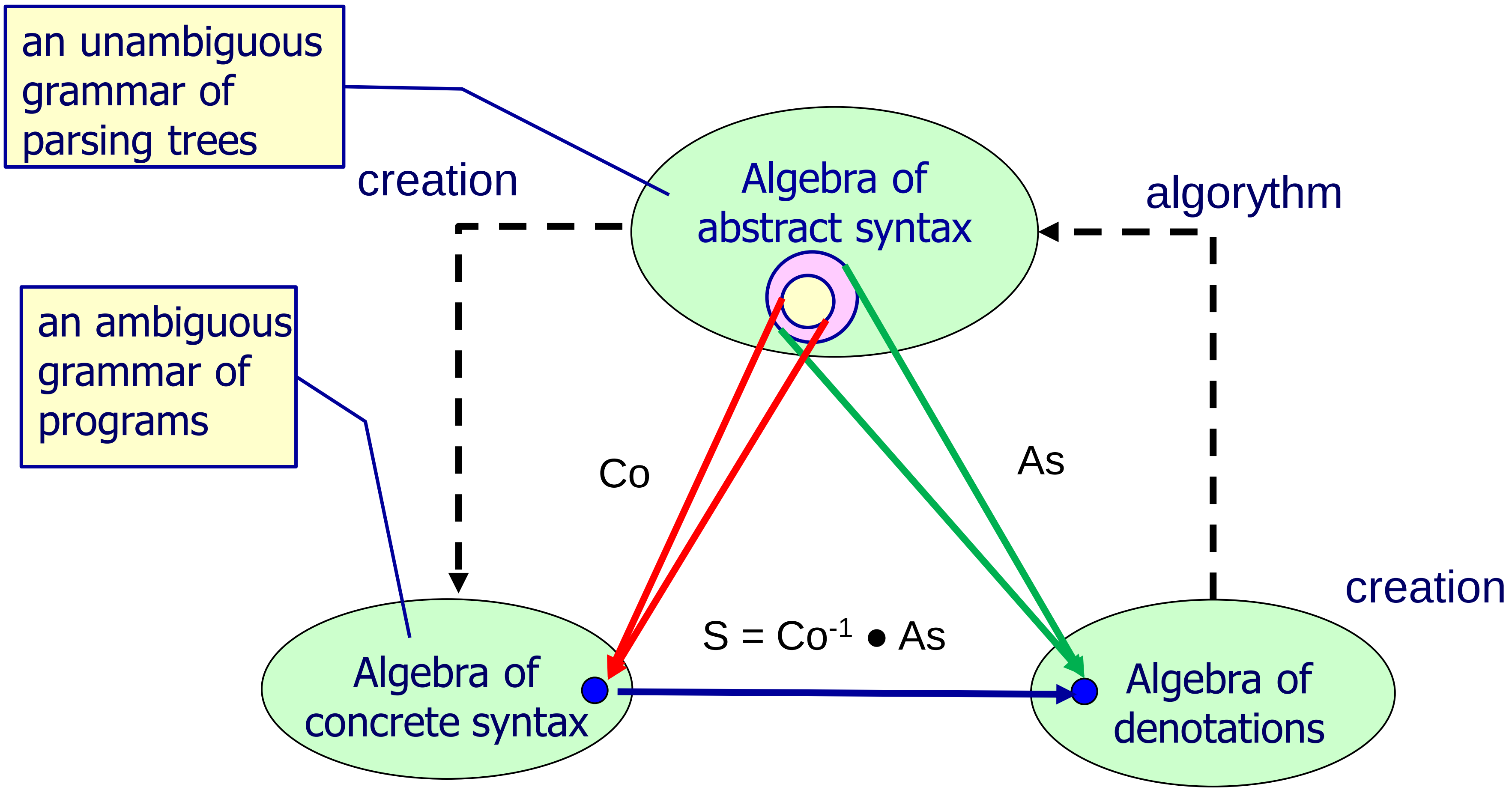
TWO SORTS OF ELEMENTS:

- numbers, e.g. real numbers
- Boolean values

REACHABLE ELEMENTS:

- {0, 1, 2,…}
- {tt, ff}

A denotational model of a programming language



If Co glues not more than As, then the (unique) homomorphism S exists.

A toy example, part 1

Carriers — Algebra of denotations

Ide = {x, y, z,…}

ExpDen = State $\to$ Number

InsDen = State $\to$ State

State = Ide $\Longrightarrow$ Number

Constructors

ide	:	$\longmapsto$ Ide for all ide : Ide
var	: Ide	$\longmapsto$ ExpDen
plus	: ExpDen x ExpDen	$\longmapsto$ ExpDen
times	: ExpDen x ExpDen	$\longmapsto$ ExpDen
assign	: Ide x ExpDen	$\longmapsto$ InsDen
compose	: InsDen x InsDen	$\longmapsto$ InsDen

Notation:

A $\to$ B   partial fun.

A $\longmapsto$ B   total fun.

A $\Longrightarrow$ B   finite fun.

ALGORITHM

Algebra (grammar) of abstract syntax

Ide = {`x, y, z`,…}

Exp = **`var`**`(`Ide`)` | **`plus`**`(`Exp, Exp`)` | **`times`**`(`Exp, Exp`)`

Ins = **`assign`**`(`Ide, Exp`)` | **`compose`**`(`Ins, Ins`)`

ALGORITHM

Semantics of abstract syntax (As)

Sid : Ide $\longmapsto$ Ide identity

Sex : Exp $\longmapsto$ ExpDen

Sin : Ins $\longmapsto$ InsDen

Semantics of abstract syntax (As)

Sid : Ide $\longmapsto$ Ide identity
Sex : Exp $\longmapsto$ ExpDen   ExpDen = State $\rightarrow$ Number
Sin : Ins $\longmapsto$ InsDen

Sex.[**plus** (Exp-1, Exp-2)] =   implementor-oriented definition
plus.[Sex.[Exp-1], Sex.[Exp-2]]

constructor of denotations

Sex.[**plus** (Exp-1, Exp-2)].sta =
Sex.Exp-i.sta = ? ➔ ? for i = 1,2
Sex.Exp-i.sta : Error ➔ Sex.Exp-i.sta for i = 1,2
true ➔ Sex.Exp-i.sta + Sex.Exp-i.sta

user-oriented definition

addition of numbers

A toy example, part 3

Algebra (grammar) of abstract syntax

Ide = {x, y, z,...}
Exp = **var**(Ide) | **plus**(Exp, Exp) | **times**(Exp, Exp)
Ins = **assign**(Ide, Exp) | **compose**(Ins, Ins)

CREATION
assisted

Algebra (grammar) of concrete syntax

Ide = {x, y, z,...}
Exp = Ide | (Exp + Exp) | (Exp * Exp)
Ins = Ide := Exp | Ins ; Ins

acceptable ambiguity

CREATION
assisted

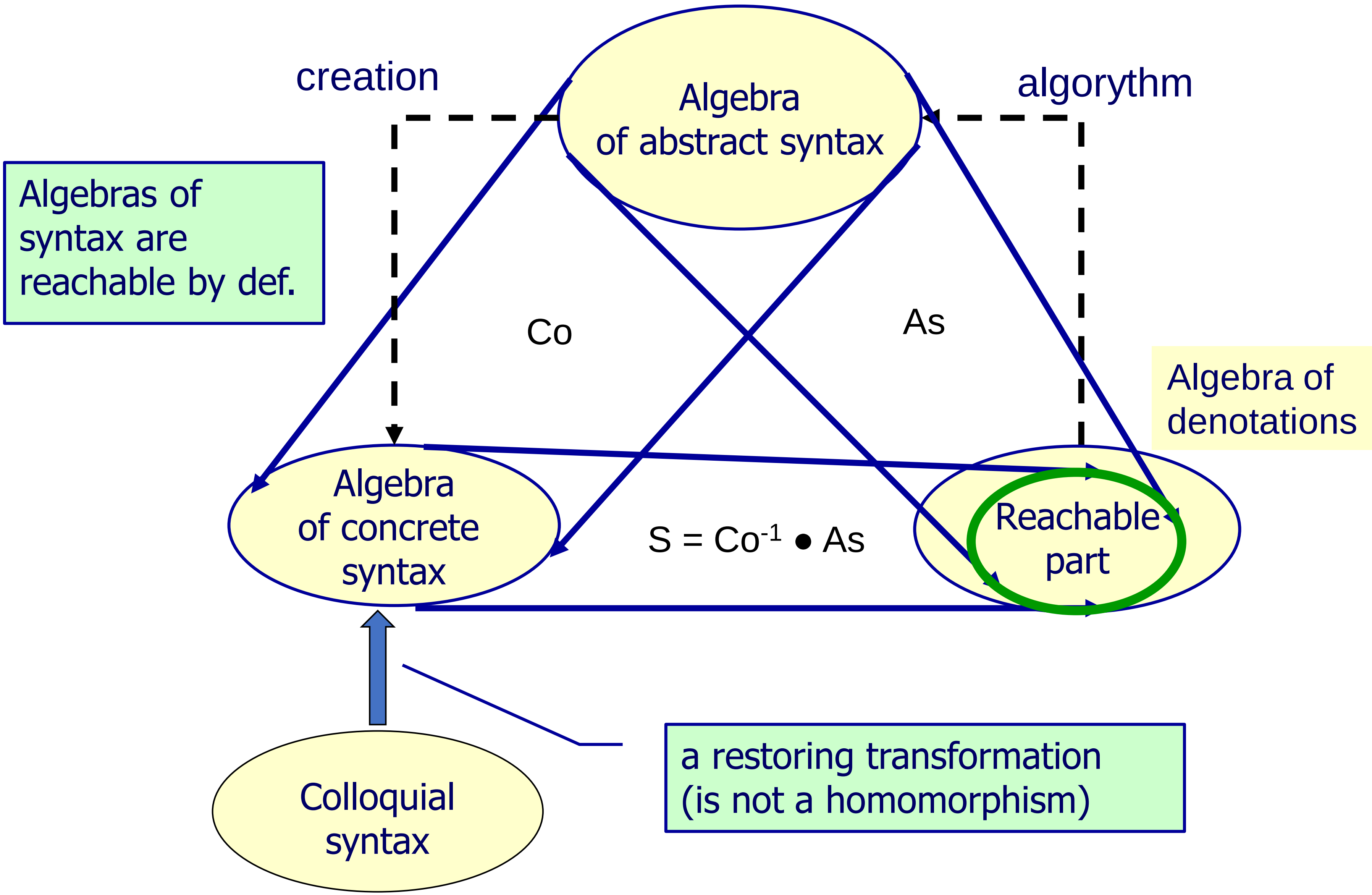
Algebra (grammar) of colloquial syntax

Ide = {x, y, z}
Exp = Ide | (Exp + Exp) | (Exp * Exp)
Exp + Exp | Exp * Exp
Ins = Ide := Exp | Ins ; Ins

not acceptable ambiguity

There is no denotational semantics for this colloquial syntax (grammar)!

A model with a colloquial syntax



Lingua – an example language
where to explain the application of our model

- Booleans, numbers, words, lists, arrays, record and their arbitrary combinations plus SQL databases;
- three-valued propositional calculus for Boolean expressions,
- abstract errors incorporated into the algebras of denotations,
- user-defined structured types,
- basic programming constructors (:=, if-then-else-fi, while-do-od)
- procedures with recursion and multirecursion,
- object-oriented programming (work in progress)
- concurrency (work in progres)
- sound program-constructors based on Hoare's logic with clean termination (three-valued predicate calculus).

What is to be done in the project

Tools – A working environment of a programmer in Lingua

1. An interpreter/compiler of Lingua; possibly developed by bootstrapping (a preliminary experimental interpreter written in Ocaml is already there)
2. An editor of programs supporting the use of sound program-construction rules
3. An adaptation of some existing theorem prover for checking conditions in the process of program development by construction rules.

What is to be done in the project

Tools – A working environment for language designer/developer

1. An editor supporting the writing of the definitions of denotation constructors.
2. A system generating abstract-syntax grammar from a signature (a meta-definition) of the algebra of denotations.
3. A system supporting the development of a concrete-syntax grammar form an abstract-syntax grammar.
4. A system supporting the generation of a restoring application from colloquial syntax into a concrete syntax.
5. A generator of semantic clauses from a concrete-syntax grammar and the definitions of denotation constructors.
6. A generator of an interpreter/compiler code from semantic clauses.

What is to be done in the project

Designing a "basic practical" Lingua-WU
(WU- stands for Warsaw University)
(partly done)

1. Formal definitions of algebras and their constructors:
   1. data-related algebras (bodies, composites, types and values)
   2. applicative denotations – data- and type expressions),
   3. imperative denotations – structured instructions and procedures
2. Grammar of concrete syntax.
3. Colloquialisms and restoration transformation.
4. A programmer's manual of Lingua-WU.
5. Some practical experiments with Lingua-WU.

What is to be done in the project

Further developments of Lingua-WU

1. The enrichment of Lingua-WU with object-oriented mechanisms; work in progress.
2. The enrichment of Lingua-WU with SQL mechanisms; work in progress.
3. The enrichment of Lingua-WU with HTML mechanisms.
4. The enrichment of Lingua-WU with tools for microprogramming.
5. A working environment of a designer of Domain Specific Languages (DSL)
6. …

What is to be done in the project

General research areas

1. A denotational model for languages with objects (work in progress)
2. A denotational model for languages with concurrency (work in progress)
3. Customer-oriented specification languages for different areas of applications.
4. Some more issues will certainly emerge in the course of the development of our model and language.


Thank you for
your attention