LINGUA

An invitation to a project

The book "A Denotational Engineering of Programming Languages" and selected research papers available on http://www.moznainaczej.com.pl/denotational-engineering

Andrzej Blikle

January 4th, 2025

Our goals

To improve the quality of programs
To lower the costs of program testing and maintenance

testing

- cost-intensive
- leaves bugs

proving correct

- proof are longer than theorems
- programs to be proved are not correct

building correct

correct by construction
programs are developed in a way that guarantees their correctness

- Edsger Dijkstra (1968, 1976),
- Andrzej Blikle (1978),
- K. Rustan M. Leino (2008) Dafny project

Bridges, airplanes, cars,... are built in this way

Two aspects of the quality of programs

It may be a future area of LINGUA project.

1. a consistency of program specification with user expectations,

2. a consistency of a program with its specification.

Current area of LINGUA project. My field of Research since 1980's.

What do we need to realize our goal?

A programming language with mathematical semantic.

We choose denotational semantics

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

Correctness preserving construction rules for programs in such a language.

They must be proved sound (correct)

Can we write a denotational semantics for an arbitrary programming language?

My hypothesis
For the majority of languages (e.g. Python, Java,…) – probably not; at least not in a direct way (see later).

For sure it hasn't been done so far!

A historical approach to defining a semantics of a language: define a semantics for a given syntax.

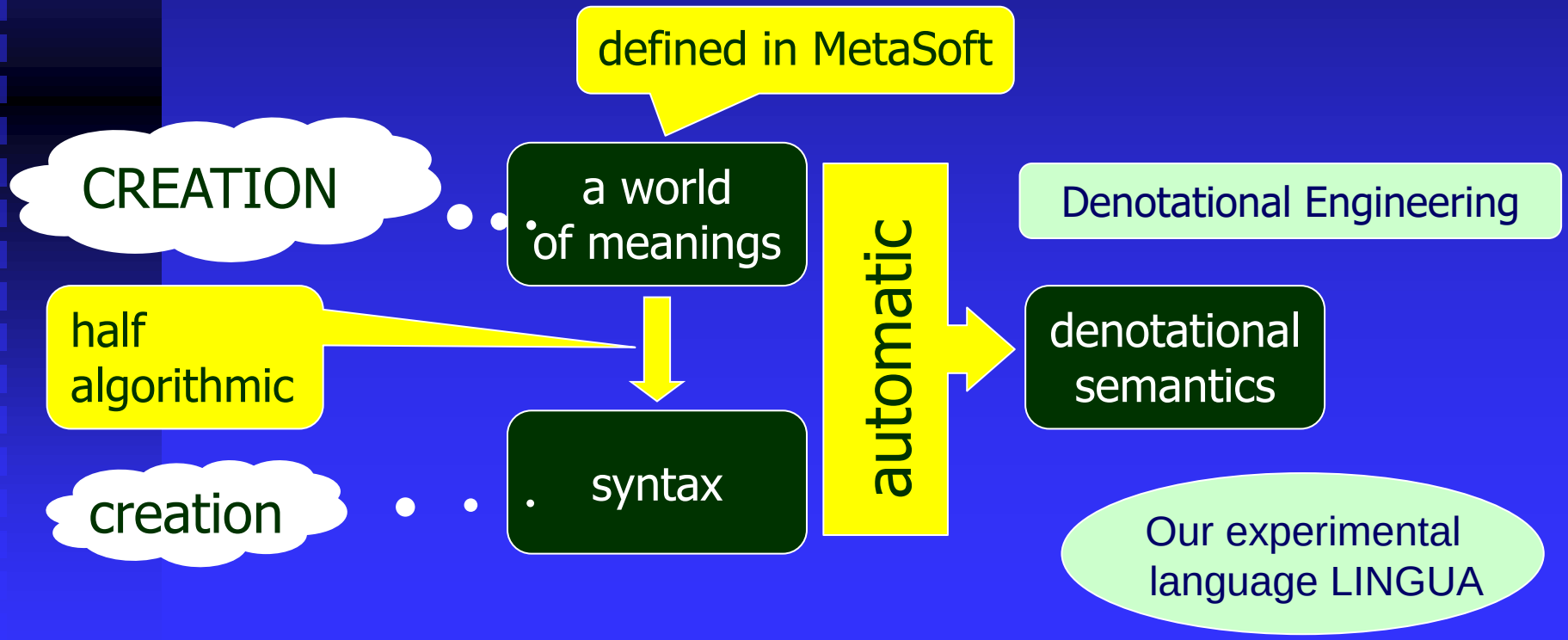
When people started to think about semantics (around 1970), syntaxes were already there!

Programs were seen as commands for computers
rather than
as descriptions of mathematical beings (semantical meanings).

Let's reverse the way from syntax to denotations

First, decide WHAT we are going to talk about:
the denotations of expressions, instructions, declarations etc.

Then, decide HOW we are going to talk about these meanings.



Where are we in the quality chain of software production?

process actors

communication languages

client

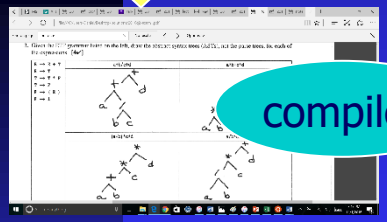
Informal contract

Our project

programmer

C++, LINGUA

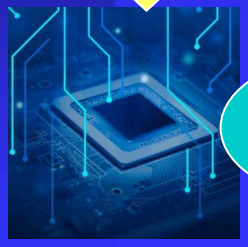
techniques of software development



compiler

Why these limitations in LINGUA?

- Our field of competence
- A lot of existing research
- Compilers, OS, and hardware have been well tested



hardware and OS

binary code

Domain Specific Languages may be built using Denotational Engineering

The state of the art of the project (a theoretical background)

a methodology of designing programming languages → from denotations to syntax

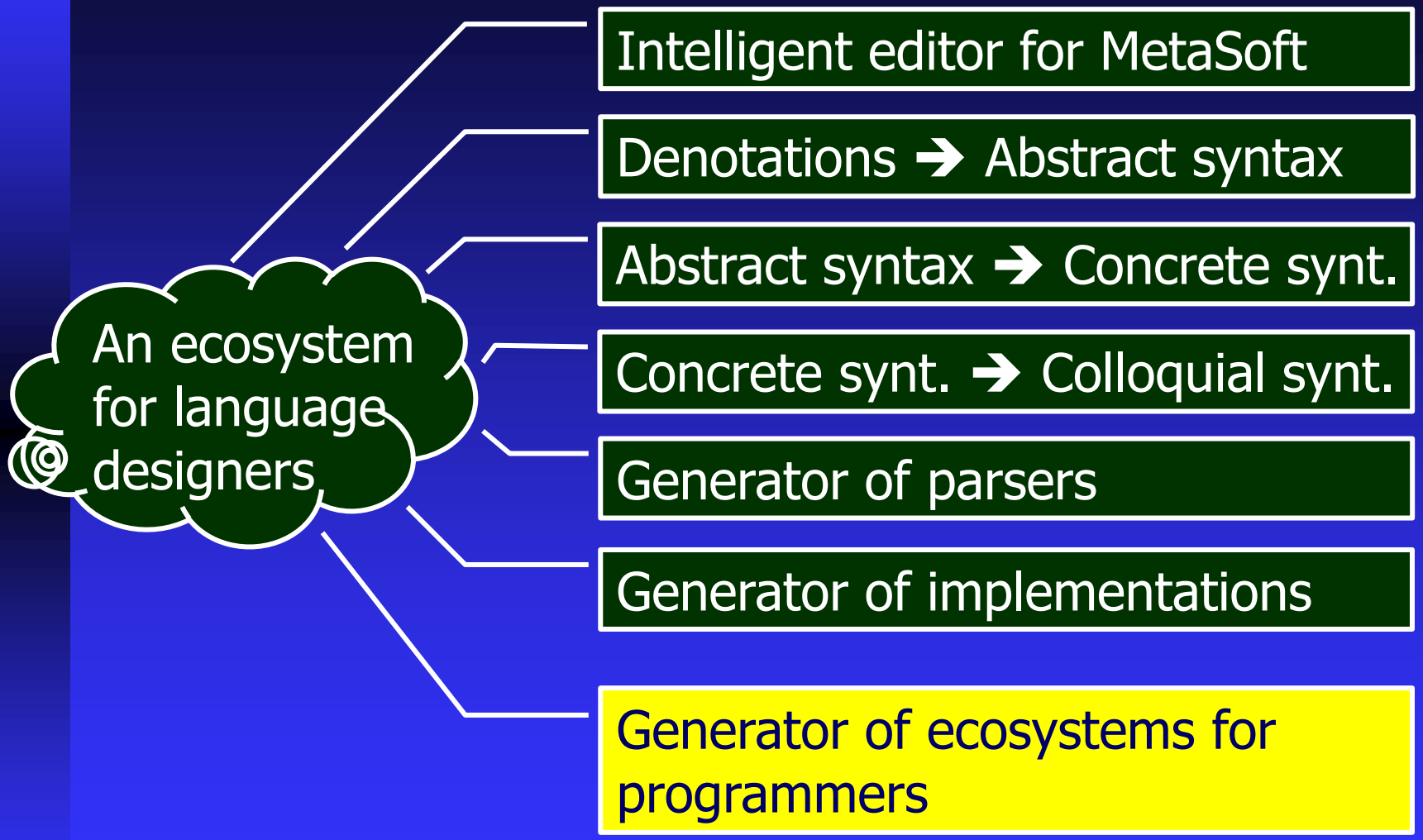
a methodology of programming → LINGUA correct by construction

An experimental interpreter

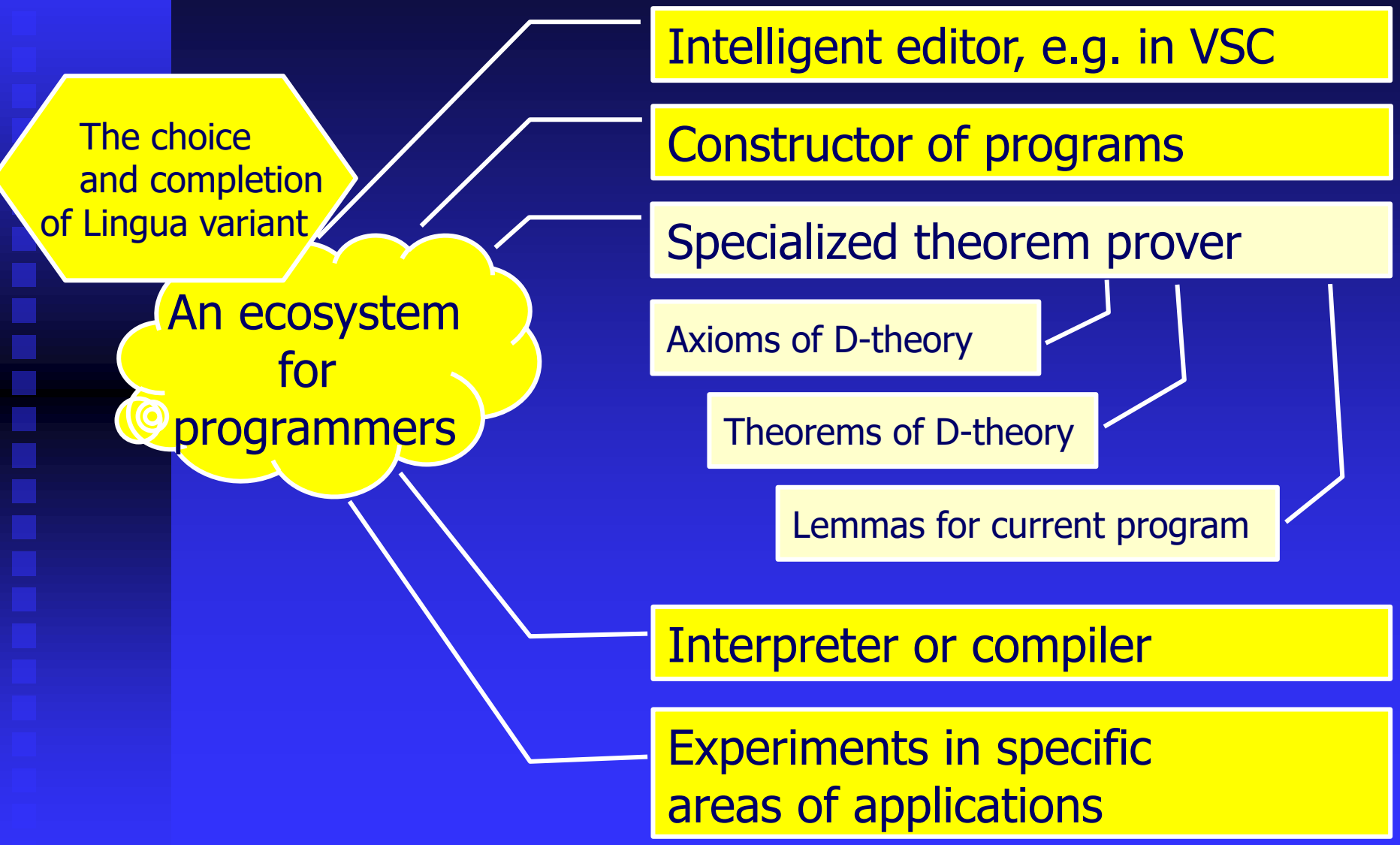
- statically typed
- object oriented
- API for SQL
- concurrency (simp. Petri nets)
- error elaboration

Documented in:
A Denotational Engineering of Programming Languages
(a book in statu nascendi)

What is to be done in software?



What is to be done in software (cont.)?



Chicken OR egg dilemma

LINGUA

An ecosystem for language designers

What remains to be done in theory (some examples)

- models for script language; HTML, TEX,…
- more about concurrency,
- polymorphic types,
- development of "practical" program-construction rules,
- domain specific languages in Lingua family; e.g., for microprogramming,
- …

A possible business model of a (future) enterprise

Currently project is being developed in a
non-institutional (informal) way
and without any budget.

My current vision of a future business model:

- LINGUA + ecosystems available free in public domain,
- open access for all but open sources for a selected group,
- monetization:
  - installation and maintenance of tools,
  - education,
  - production of reliable software,
  - …

A TOY EXAMPLE OF A LANGUAGE DESIGN

Our method bases on many-sorted algebras

BAD NEWS
This theory is technically a little complicated.

GOOD NEWS
You do not need to master it very deeply.

Preliminary notations
MetaSoft

Sets in the theory of denotational semantics are traditionally called domains.

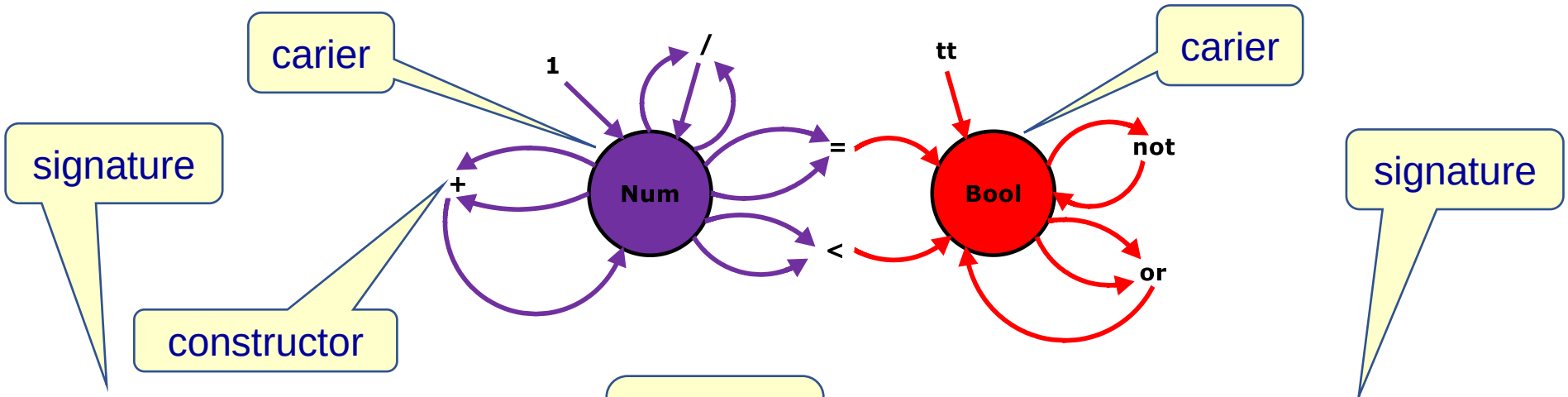
Sets (domains) and functions

A x B	the Cartesian product of sets A and B
A | B	the union of sets A and B
a : A	element a belongs to set A
A → B	the set of all partial functions from A to B
A ↦ B	the set of all total functions from A to B
A ⇒ B	the set of all finite functions from A to B
f : A → B	function f is a partial function from A to B

Abstract errors

Errors = {'division by zero', 'types not compatible', 'variable not declared, …}
DomainE = Domain | Error
elm : Domain = … elm (element) is a metavariable running over Domain
f.a.b.c = ((f.a).b).c

Many-sorted algebras intuitively



The algebra of denotations

similar algebras

The algebra of syntax

1	:	⟼ NumE	1	:	⟼ NumExp
+	: NumE x NumE	⟼ NumE	+	: NumExp x NumExp	⟼ NumExp
/	: NumE x NumE	⟼ NumE	/	: NumExp x NumExp	⟼ NumExp
=	: NumE x NumE	⟼ BoolE	=	: NumExp x NumExp	⟼ BoolExp
<	: NumE x NumE	⟼ BoolE	<	: NumExp x NumExp	⟼ BoolExp
tt	:	⟼ BoolE	tt	:	⟼ BoolExp
not	: BoolE	⟼ BoolE	not	: BoolExp	⟼ BoolExp
or	: BoolE x BoolE	⟼ BoolE	or	: BoolExp x BoolExp	⟼ BoolExp

The reachable elements of Num are all positive rational numbers

All elements of NumExp are(by def.) reachable and are expressions like: 1+(1/(1+1))

From denotations to syntax and semantics

A toy example, part 1

Carriers

Ide = {x, y, z,…}

ExpDen = State → NumE

InsDen = State → State

Algebra of denotations

State = Ide ⟹ Num

Constructors

ide	:		⟼ Ide for all ide : Ide
var	: Ide		⟼ ExpDen
plus	: ExpDen	x ExpDen	⟼ ExpDen
divide	: ExpDen	x ExpDen	⟼ ExpDen
assign	: Ide	x ExpDen	⟼ InsDen
compose	: InsDen	x InsDen	⟼ InsDen

ALGORITHM

Algebra (grammar) of abstract syntax

Ide = {x, y, z,…}

Exp = **var**(Ide) | **plus**(Exp, Exp) | **divide**(Exp, Exp)

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

ALGORITHM

Semantics of abstract syntax (As)

Sid : Ide ⟼ Ide identity

Sex : Exp ⟼ ExpDen

Sin : Ins ⟼ InsDen

The unique (!) semantics of abstract syntax

A toy example, part 2

Sid : Ide ⟼ Ide identity
Sex : Exp ⟼ ExpDen ExpDen = State → NumE
Sin : Ins ⟼ InsDen InsDen = State → State

Sex.[divide(Exp-1, Exp-2)] =
  divide.[Sex.[Exp-1], Sex.[Exp-2]]

implementor-oriented definition

constructor of denotations

Sex.[divide(Exp-1, Exp-2)].sta =
  Sex.Exp-2.sta = 0 ➔ 'division-by-zero'
  true ➔ Sex.Exp-1.sta / Sex.Exp-2.sta

arithmetical operation
(from implementation platform)

programmer-oriented definition

From abstract to colloquial syntax

A toy example, part 3

Algebra (grammar) of abstract syntax

Ide = {x, y, z,…}
Exp = **var**(Ide) | **plus**(Exp, Exp) | **divide**(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 (associativity)

CREATION assisted

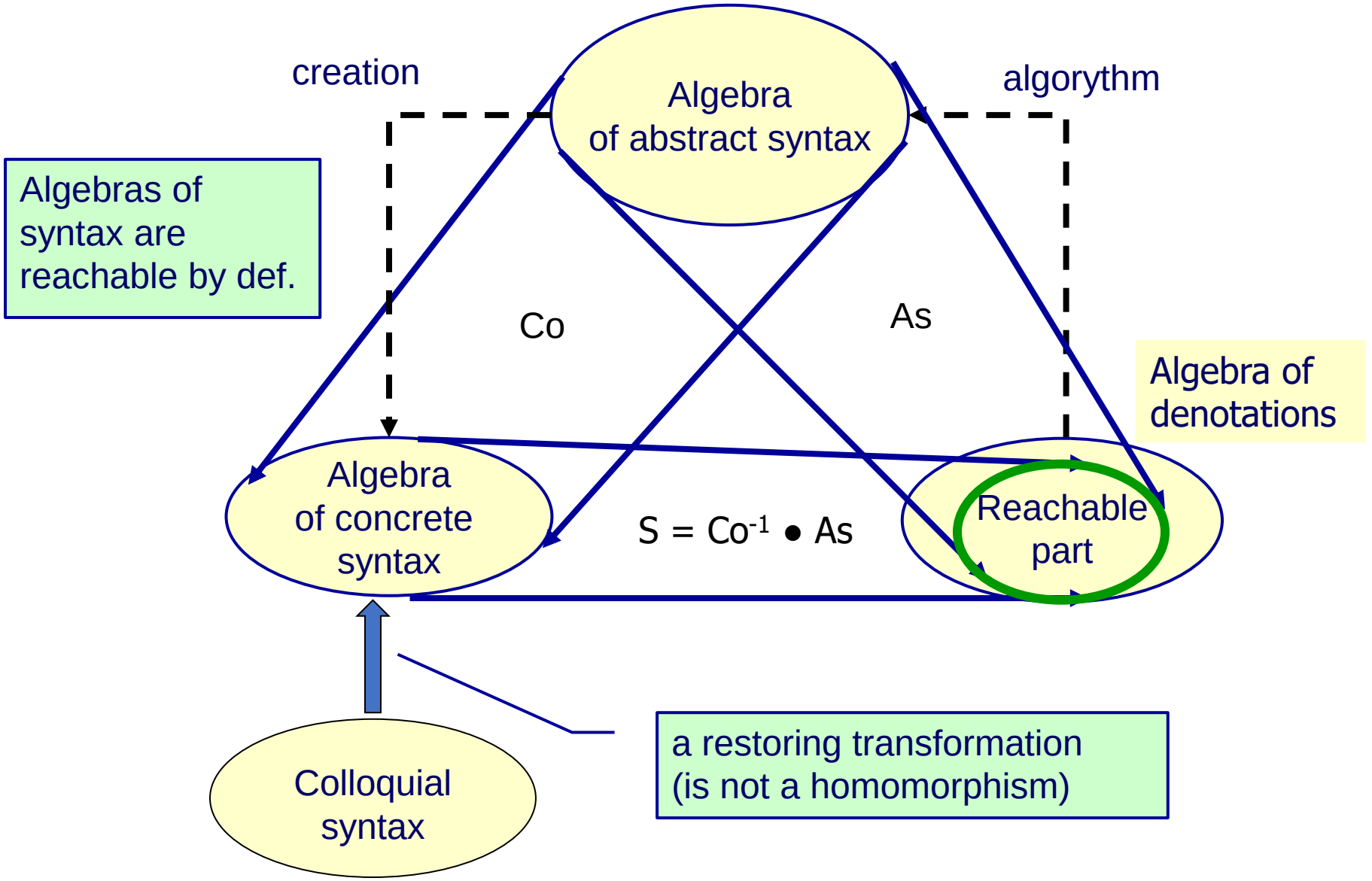
Algebra (grammar) of colloquial syntax

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



A TOY EXAMPLE OF A PROGRAM DEVELOPMENT

Installing an appliance on an engine

Step 1: A trivial search engine
(linear search for number k)

```
pre x,k is nnint :
  x := 0;
  while x+1 ≤ k
    do x := x+1 od
post x = k
```

Def: $isrt(n)^2 \le n < (isrt(n)+1)^2$

Step 2: A trivial program

```
pre x,n is nnint :
  x := 0;
  while x+1 ≤ isrt(n)
    do x := x+1 od
post x = isrt(n)
```

installing an appliance

```
x+1 ≤ isrt(n) ≡ (x+1)^2 ≤ n whenever x,n are nnint
```

Step 2: a slow program

```
pre x,n is nnint :
  x := 0;
  asr x,n is nnint
  while (x+1)^2 ≤ n
    do x := x+1 od
  rsa
post x = isrt(n)
```

If we wish to speed up our program, we have to change the engine.

The derivation of Dahl's integer square root (1)

(deriving a logarithmic search engine)

The magnitude of k: If $2^m \le k < 2^{m+1}$ then mag.k = 2^m e.g. mag.11 = 8

Def: po2.k iff $(\exists m \ge 0)\ k = 2^m$: k is a power of 2

Q1: **pre** x,k,z **is** nnint : searches for 2*mag.k e.g. 2*mag.11 = 16

```
    z := 1;
    asr x,k,z is nnint and po2.z :
      while z ≤ k do z:=2*z od
    rsa
  post x,k,z is nnint and z = 2*mag.k
```

combine these programs sequentially

Q2: **pre** x,k,z **is** nnint **and** z = 2*mag.k:

```
    x := 0;
    while z > 1
      do
        z := z/2;
        if x+z ≤ k then x:=x+z fi
      od
A post x = k and z = 1
```

k = 11

2*mag.11 = 16

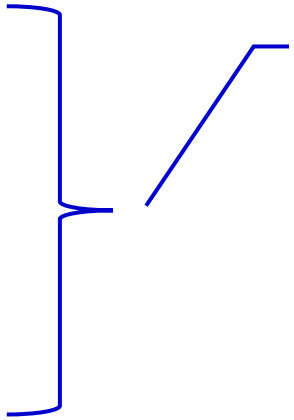
11 = 1*8 + 0*4 + 1*2 + 1*1

The derivation of Dahl's integer square root (2)

(with a logarithmic search engine)

Q3: **pre** x,k,z **is** nnint : a "pure" search engine

```
    z := 1;
    x := 0;
     asr x,k,z is nnint and po2.z :
        while z ≤ k do z:=2*z od ;
        while z > 1
          do
            z := z/2;
            if x+z ≤ k then x:=x+z fi
          od
      rsa
    post x = k and z = 1
```



the range of an assertion

Replace k by isrt(n) and use

$z \le isrt(n) \equiv z^2 \le n$ **whenever** z,n **is** nnint

$x+z \le isrt(n) \equiv (x+z)^2 \le n$ **whenever** z,n,x **is** nnint

A

The derivation of Dahl's integer square root (3)

(with a logarithmic search engine)

Q4:

```
pre z,x,n is nnint:
    z := 1;
    x := 0
    asr z,x,n is nnint and po2.z :
      while z² ≤ n do z:=2*z od
      while z > 1
        do
          z := z/2;
          if (x+z)² ≤ n then x:=x+z fi
        od
    rsa
post x = isrt(n) and z = 1
```

We shall optimize this program by eliminating both square operations.

First introduce new variable q with $q=z^2$ to avoid the recalculation of z^2

The derivation of Dahl's integer square root (11)

(with a logarithmic search engine)

Q10:

```
pre n, q, y, p is nnint:
  q :=  1;
  while q ≤ n do q:=4*q od
  y:= n;
  p:= 0;
  while q > 1
    do
      q:=q/4;
      if p+q ≤ y
        then p:=p+q; y:=y-p-q
        else p:=p/2
      fi
    od
post p=isrt(n)
```

The used operations are easily implementable in binary arithmetic.

This is the Ole Dahl's program.

THANK YOU FOR YOUR PATIENCE

You can write to me:
andrzej.blikle@moznainaczej.com.pl