

A Denotational Engineering of Programming Languages

...

Part 2: Many-sorted algebras
(Sections 2.10 – 2.14 of the book)

Andrzej Jacek Blikle

March 3rd, 2020

Many-sorted algebras

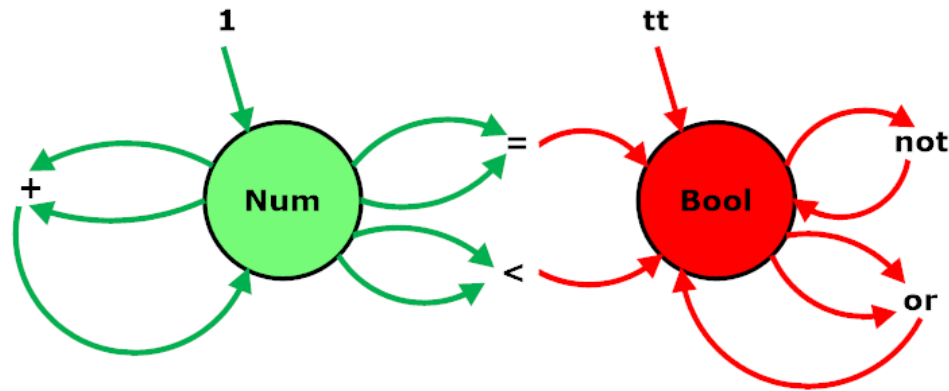
A BAD NEWS

This theory is technically a bit complicated.

A GOOD NEWS

You do not need to master it very deeply.

Many-sorted algebras intuitively



signature

signature

The algebra of data
NumBool

The algebra of expressions
NumBoolExp

1	:		\mapsto	NumE
+	:	NumE x NumE	\mapsto	NumE
*	:	NumE x NumE	\mapsto	NumE
=	:	NumE x NumE	\mapsto	BoolE
<	:	NumE x NumE	\mapsto	BoolE
tt	:		\mapsto	BoolE
not	:	BoolE	\mapsto	BoolE
or	:	BoolE x BoolE	\mapsto	BoolE

1	:		\mapsto	NumExp
+	:	NumExp x NumExp	\mapsto	NumExp
*	:	NumExp x NumExp	\mapsto	NumExp
=	:	NumExp x NumExp	\mapsto	BoolExp
<	:	NumExp x NumExp	\mapsto	BoolExp
tt	:		\mapsto	BoolExp
not	:	BoolExp	\mapsto	BoolExp
or	:	BoolExp x BoolExp	\mapsto	BoolExp

Due to abstract errors all functions (in this case) can be made total

Abstract and concrete syntax

Algebra of data AlgNumBool repeated

1	:		\mapsto	NumExp	<	:	NumExp x NumExp	\mapsto	BoolExp
+	:	NumExp x NumExp	\mapsto	NumExp	tt	:		\mapsto	BoolExp
*	:	NumExp x NumExp	\mapsto	NumExp	not	:	BoolExp	\mapsto	BoolExp
=	:	NumExp x NumExp	\mapsto	BoolExp	or	:	BoolExp x BoolExp	\mapsto	BoolExp

Abstract syntax Prefix notation: **not (< (+ (1, * (1, 1)), + (1, 1)))**

NumExp = **1** | **+** (NumExp, NumExp) | ***** (NumExp, NumExp)

BoolExp = **tt** | **=** (NumExp, NumExp) | **<** (NumExp, NumExp) |
not (BoolExp) | **or** (BoolExp, BoolExp)

Concrete syntax Infix notation: **not ((1+(1*1)) < (1+1))**

NumExp = **1** | (NumExp **+** NumExp) | (NumExp ***** NumExp)

BoolExp = **tt** | (NumExp **=** NumExp) | (NumExp **<** NumExp) |
not (BoolExp) | (BoolExp **or** BoolExp)

Abstract, concrete and colloquial syntax

Abstract syntax – algorithmically derivable from algebra's signature

`not (< (+ (1, * (1, 1) , + (1, 1))))`

no creativity

Concrete syntax – an isomomorphic transformation: abstract \rightarrow concrete

`not ((1+ (1*1)) < (1+1))`

creativity

Colloquial syntax – a restoring transformation: concrete \leftarrow colloquial

`not (1+1*1 < 1+1)` — here the omission of "unnecessary" parentheses

Colloquial assumptions

creativity

* binds stronger than +

* and + bind stronger than <

Denotational semantics of concrete syntax

A two-sorted homomorphism

$SN : \text{NumExp} \mapsto \text{NumE}$

$SB : \text{BooExp} \mapsto \text{BooE}$

Semantic clauses (two examples):

$SN[1] = 1$

$SN[\text{exp-1} + \text{exp-2}] =$

$SN[\text{exp-1}] + SN[\text{exp-2}] \geq \text{max} \rightarrow$ 'overflow'

true

$\rightarrow SN[\text{exp-1}] + SN[\text{exp-2}]$

The meaning of a whole
is a combination of the
meanings of its parts

max — maximal number for a current implementation

Many-sorted algebras formally

$\underline{\text{Alg}} = (\text{Sig}, \text{Car}, \text{Fun}, \text{car}, \text{fun})$ — algebra
 $\text{Sig} = (\text{Cn}, \text{Fn}, \text{ar}, \text{so})$ — signature

Cn — finite set of words; carrier names

Fn — finite set of words; function names

$\text{ar} : \text{Fn} \mapsto \text{Cn}^{c^*}$ — arity

$\text{so} : \text{Fn} \mapsto \text{Cn}$ — sort

Car — a finite family of sets called carriers

Fun — a finite family of function called constructors

$\text{car} : \text{Cn} \mapsto \text{Car}$

$\text{fun} : \text{Fn} \mapsto \text{Fun}$

Similarity and homomorphism

similar algebras — have the same signature

an extension of \underline{Alg} results from \underline{Alg} by adding:

- new carriers, and/or
- new elements to the existing carriers, and/or
- new function

$\underline{Alg}_i = (\text{Sig}, \text{Car}_i, \text{Fun}_i, \text{car}_i, \text{fun}_i)$ for $i = 1, 2$ — similar algebras
 $\text{Sig} = (\text{Cn}, \text{Fn}, \text{ar}, \text{so})$ — common signature

\underline{Alg}_1 is a subalgebra of similar \underline{Alg}_2 if

- $\text{car}_1.\text{cn} \subset \text{car}_2.\text{cn}$ for any $\text{cn} : \text{Cn}$
- constructors of Fun_1 coincide with the corresponding constructors of Fun_2 on their domains

a homomorphism $H : \underline{Alg}_1 \mapsto \underline{Alg}_2$, $H = \{h.\text{cn} \mid \text{cn} : \text{Cn}\}$

$$h.\text{cn}.\text{fun}_1.\text{fn}.(a_1, \dots, a_n) = \text{fun}_2.\text{fn}.(h.\text{cn}_1.a_1, \dots, h.\text{cn}_n.a_n)$$

Reachable algebras and abstract syntax

reachable subalgebra — all elements constructible by constructors

reachable algebra — identical to its (unique) reachable subalgebra

$\text{Int} = (\text{Integer}, 1, +)$ is a reachable subalgebra of $\text{Num} = (\text{Number}, 1, +)$

abstract syntax over Sig a reachable algebra denoted $\text{AbsSyn}(\text{Sig})$

$\text{Sig} = (\text{Cn}, \text{Fn}, \text{ar}, \text{so})$

carriers — formal languages over $\text{Alphabet} = \text{Fn} \mid \{ (,), \dots \}$

with every $\text{fn} : \text{Fn}$ we assign a constructor of languages

$+ : \text{Integer} \times \text{Integer} \mapsto \text{Integer}$ — a constructor of numbers

$[+] : \text{IntExp} \times \text{IntExp} \mapsto \text{IntExp}$ — a corresponding constructor of expr.

$[+].(\text{exp}_1, \text{exp}_2) = '+' \circ (' \circ \text{exp}_1 \circ ', \circ \text{exp}_2 \circ ')'$

$= +(\text{exp}_1, \text{exp}_2)$ — a simplified notation for constructors

equational grammar:

$\text{IntExp} = \{1.\} \circ \{ \{ (\circ \{ \}) \} \mid$

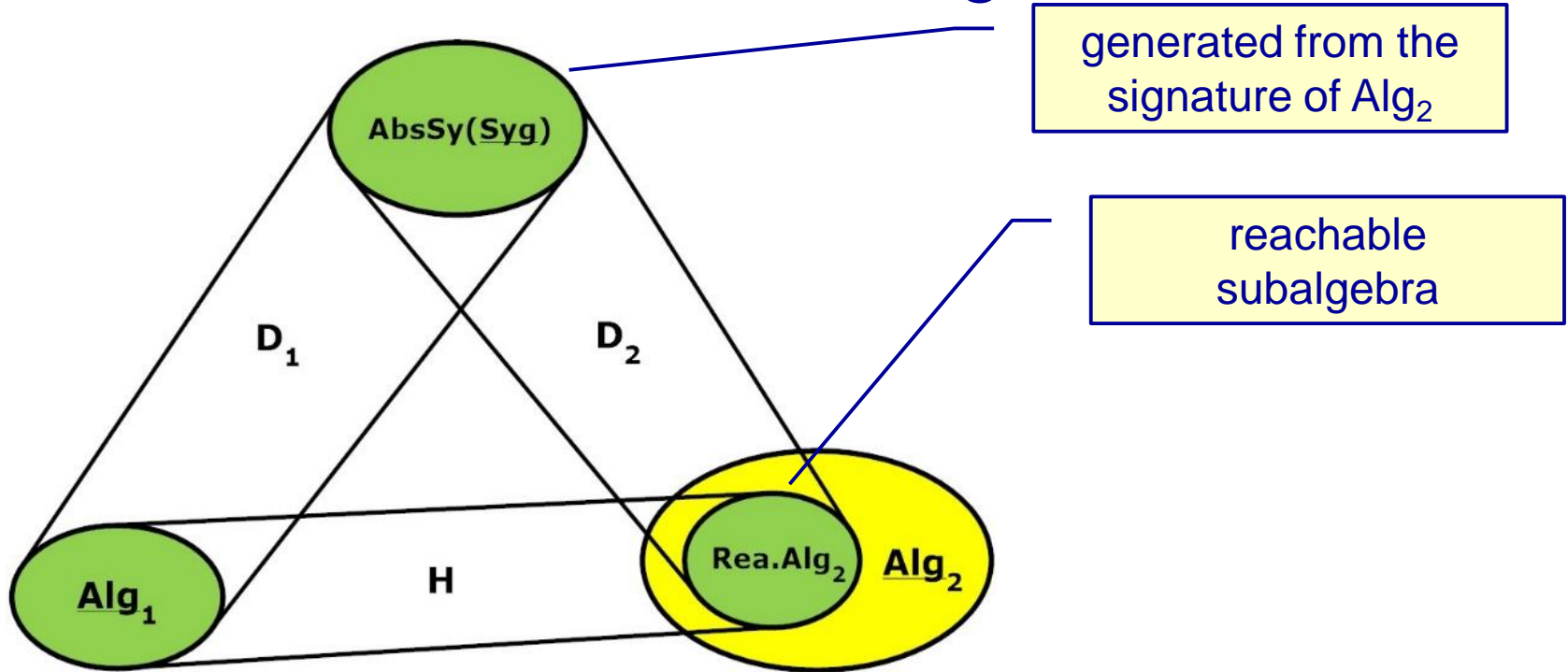
$\{+\} \circ \{ \{ (\circ \text{IntExp} \circ \{,\} \circ \text{IntExp} \circ \{ \}) \}$

$= 1 \mid$

$+(\text{IntExp}, \text{IntExp})$

— a simplified notation for grammars

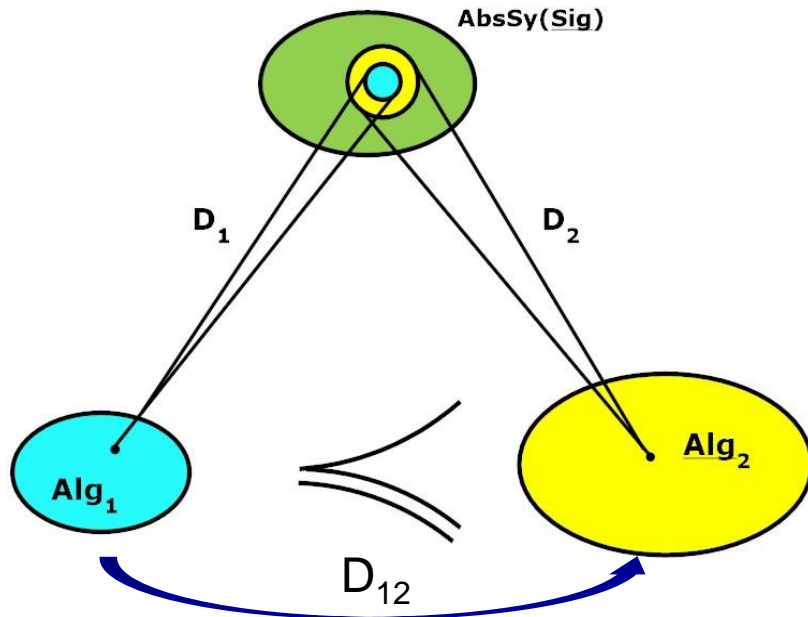
Two facts about algebras



For every \underline{Alg} with Sig there is exactly one homomorphism
 $D_2 : \text{AbsSyn}(\text{Sig}) \mapsto \underline{Alg}$

If \underline{Alg}_1 and \underline{Alg}_2 are similar and \underline{Alg}_1 is reachable then there is at most one homomorphism
 $H : \underline{Alg}_1 \mapsto \underline{Alg}_2$ $(H : \underline{Alg}_1 \mapsto \text{Reachable}.\underline{Alg}_2)$

Ambiguous and unambiguous algebras



An algebra is called ambiguous, if its unique homomorphism from abstract syntax is not a one-one homomorphism (if it is gluing)

Algebra \underline{Alg}_1 is said to be not more ambiguous than \underline{Alg}_2 , if D_1 is gluing not more than D_2 .

If

- \underline{Alg}_1 and \underline{Alg}_2 are similar (have a common signature) and
- \underline{Alg}_1 is reachable,

then

the (unique) homomorphism $D_{12} : \underline{Alg}_1 \mapsto \underline{Alg}_2$ exists iff $\underline{Alg}_1 \preceq \underline{Alg}_2$.

If D_1 is an isomorphism then \underline{Alg}_1 is unambiguous, and the (unique) homomorphism $D_{12} : \underline{Alg}_1 \mapsto \underline{Alg}_2$ exists ($D_{12} = D_1^{-1} \bullet D_2$)

Syntactic algebras versus grammars

An algebra is called a **syntactic algebra**, if it is a reachable algebra of words.

DEF A **skeleton function**: $f.(x_1, \dots, x_k) = w_1 x_1 \dots w_k x_n w_{k+1}$.
(w_1, \dots, w_k, w_{k+1}) — skeleton

$F.(exp-b, ins_1, ins_2) = \text{if } exp-b \text{ then } ins_1 \text{ else } ins_2 \text{ fi}$ — F is skeleton f.
 $F.(exp-b, ins_1, ins_2) = \text{if } exp-b \text{ then } ins_2 \text{ else } ins_1 \text{ fi}$ — F is not skeleton f

DEF A **context-free algebra** – all its constructors are skeleton functions

For every context-free algebra there is an equational grammar that generates its carriers.

For every equational grammar there is a context-free algebra with carriers defined by that grammar

Colloquial syntax versus traditional approach

CONCRETE SYNTAX

ConExp = 1 | (ConExp + ConExp) |
(ConExp * ConExp)

Parentheses are
optional

COLLOQUIAL SYNTAX

ColExp = 1 | (ColExp + ColExp) |
ColExp + ColExp |
(ColExp * ColExp) |
ColExp * ColExp

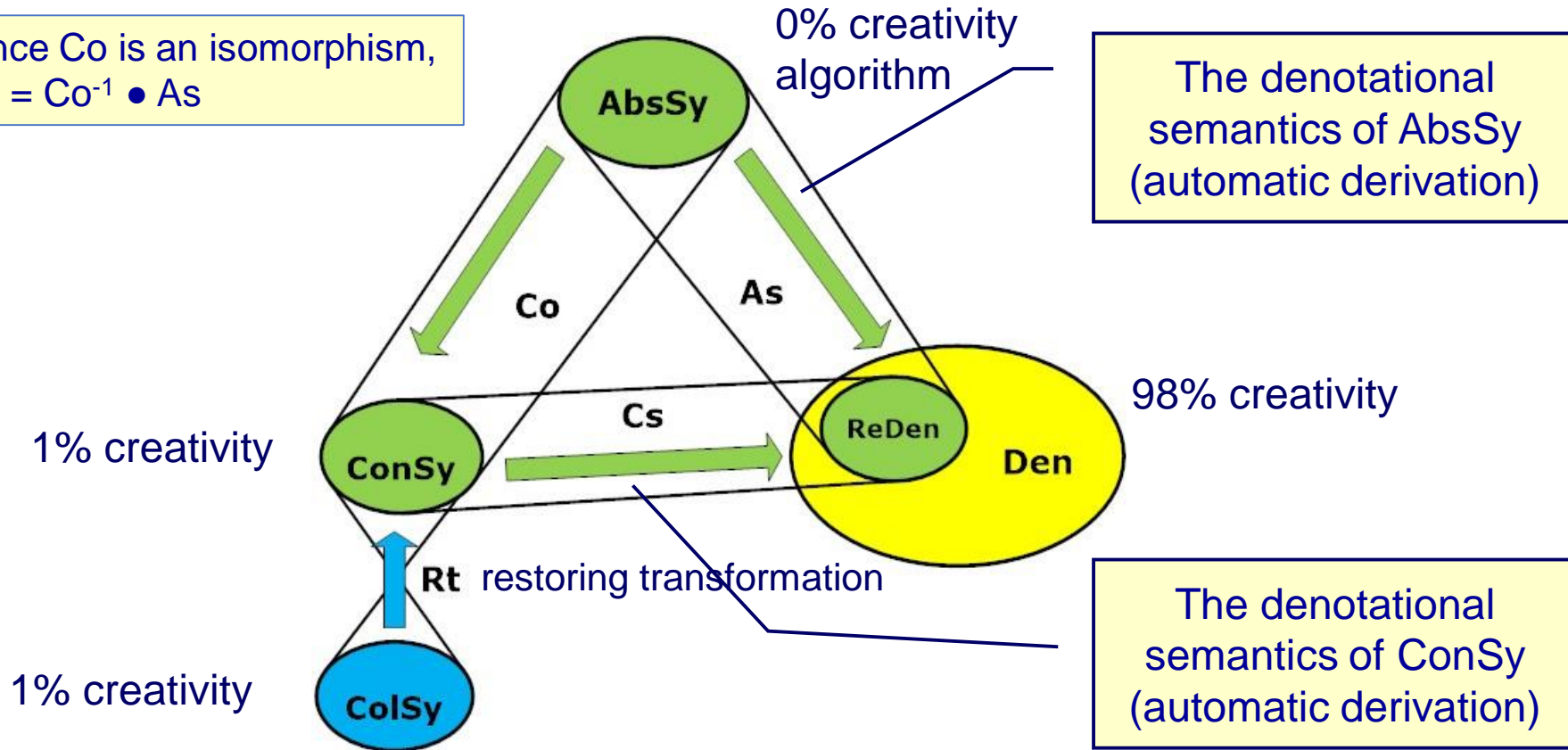
This requires a redefinition of
our algebra of denotations

TRADITIONAL APPROACH

Expression = Component | Expression + Component
Component = Factor | Factor * Component
Factor = 1 | (Expression)

A recapitulation of an algebraic model of a programming language

Since Co is an isomorphism, $Cs = Co^{-1} \bullet As$



Two steps of program execution:

1. restoring transformation
2. interpretation/compilation based on Cs .



Thank you for
your attention