

# A Denotational Engineering of Programming Languages

...

Part 1: MetaSoft, CPO's and McCarthy's propositional calculus  
(Sections 2.1 – 2.9 of the book)

Andrzej Jacek Blikle

March 3<sup>re</sup>, 2020



"Denotational Engineering of Programming Languages" by Andrzej Blikle is licensed under a Creative Commons:  
Attribution — NonCommercial — NoDerivatives.

# MetaSoft

A definitional metalanguage for  
denotational definitions of  
programming languages

Developed in  
Institute of Computer Science  
Polish Academy of Sciences  
in the years 1970 - 1980

# MetaSoft notation — functions

$a : A$	— $a$ is an element of $A$
$\{a_1, \dots, a_n\}$	— a finite set
$\{\}$	— an empty set
$f.a$	— $f(a)$ ,
$f.a.b.c = ((f(a))(b))(c)$	— (Haskell) Curry's notation
$f.a = !, f.a = ?$	— $f$ is defined/undefined for $a$
$(f \bullet g).a = g.(f.a)$	— sequential composition of functions
$f = [a_1/b_1, \dots, a_n/b_n]$	— mapping: $f.a_i = b_i$ and undefined otherwise
$[\ ]$	— empty function
$[A].a = a$	— identity function; where $A$ is a set
$f : A \rightarrow A$	— a (possibly partial) function from $A$ to $A$
$f^0 = [A]$	— 0-th iteration
$f^n = f^{n-1} \bullet f$	— $n$ -th iteration

# MetaSoft notation — functions (cont.)

$f : A \rightarrow B, \quad g : C \rightarrow D$

$f \blacklozenge g : A|C \rightarrow B|D$  —  $f$  overwritten by  $g$

$(f \blacklozenge g).x =$

$g.x = ! \quad \rightarrow g.x$  if  $g.x$  defined then  $g.x$

$g.x = ? \quad \rightarrow f.x$  if  $g.x$  undefined then  $f.x$

$f \blacklozenge g = f | g$  — if  $f \cap g = \{ \}$

$f[a_1/b_1, \dots, a_n/b_n]$  —  $f$  overwritten by a mapping  $[a_1/b_1, \dots, a_n/b_n]$

$f[a_1/b_1, \dots, a_n/b_n].x =$

$x = a_1 \rightarrow b_1$  if  $x = a_1$  then,  $f.x = b_1$ , and otherwise

...

$x = a_n \rightarrow b_n$  if  $x = a_n$  then,  $f.x = b_n$ , and otherwise

**true**  $\rightarrow f.x$  in all other cases  $f.x$

# MetaSoft notation: sets (domains)

$\{a_1, \dots, a_n\}, \{\}$	— a <u>finite set</u> , an <u>empty set</u>
$\text{Sub}.A, \text{FinSub}.A$	— families of all (all finite) subsets of $A$ ,
$A \mid B$	— union of $A$ and $B$ ,
$A \times B$	— Cartesian product of $A$ and $B$ ,
$A^{c^*}$	— all finite (or empty) tuples $(a_1, \dots, a_n); a_i : A$
$()$	— empty tuple
$A^{c^+}$	— all finite non-empty tuples on $A$
$A \rightarrow B$	— all partial functions from $A$ to $B$
$A \mapsto B$	— all total function from $A$ to $B$
$A \Rightarrow B$	— all mappings (finite functions) from $A$ to $B$

$\text{Rel}(A, B) = \{\text{rel} \mid \text{rel} \subseteq A \times B\}$  — binary relations

typical domain equations:

$\text{val} : \text{Value} = \text{Data} \times \text{Type}$

$\text{vat} : \text{Valuation} = \text{Identifier} \Rightarrow \text{Value}$



vat – a metavariable running over Valuation

# Chain-complete partially ordered sets

$\sqsubseteq : \text{Rel}(A,A) = \{R \mid R \subseteq A \times A\}$       ordering relation in  $A$

DEF. partial order:

$a \sqsubseteq a$       reflexivity  
if  $a \sqsubseteq b$  and  $b \sqsubseteq c$  then  $a \sqsubseteq c$       transitivity  
if  $a \sqsubseteq b$  and  $b \sqsubseteq a$  then  $a = b$       weak antisymmetry

$b : B$  is called the least element in  $B \subseteq A$  if  $(\forall b' : B) b \sqsubseteq b'$

$a : A$  is called the upper bound of  $B \subseteq A$ , if  $(\forall b : B) b \sqsubseteq a$

$a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \dots$       a chain  
 $\text{lim}(a_i \mid i = 1, 2, \dots)$       limit = least upper bound (if exists)

Def.  $(A, \sqsubseteq, \Phi)$  is called chain-complete partially ordered set (CPO) if:

1. every chain in  $A$  has a limit,
2.  $\Phi$  is the least element of  $A$

# Continuous functions in CPO's

$(A, \sqsubseteq, \Phi)$  — CPO

DEF  $f : A \mapsto A$  is continuous if

1. if  $a_1 \sqsubseteq a_2 \sqsubseteq \dots$  then  $f.a_1 \sqsubseteq f.a_2 \sqsubseteq \dots$ ,
2. if  $a_1 \sqsubseteq a_2 \sqsubseteq \dots$  has a limit then  $f.a_1 \sqsubseteq f.a_2 \sqsubseteq \dots$  has a limit,
3.  $\lim(f.a_1 \sqsubseteq f.a_2 \sqsubseteq \dots) = f.[\lim(a_1 \sqsubseteq a_2 \sqsubseteq \dots)]$ .

A composition of continuous functions is continuous.

Kleene's fixed-point theorem

A fundament for recursive definitions of function and domains

If  $f : A \mapsto A$  is continuous, then the least solution of

$$x = f.x$$

exists and equals  $\lim(f^n.\Phi \mid n = 0, 1, 2, \dots)$ .

DEF A set-theoretic CPO — A CPO of sets with ordering by inclusion and the empty set  $\{ \}$  as the least element. E.g. the Sub.A.

# A fixed-point definition of $2^n$

$\text{Nat} = \{0, 1, 2, \dots\}$  — natural numbers  
 $(\text{Nat} \rightarrow \text{Nat}, \subseteq, [ \ ])$  — a set-theoretic CPO of partial functions on  $\text{Nat}$   
 $f \bullet g$  — continuous on both arguments  
 $f \blacklozenge g$  — continuous on both arguments

$\text{power} : \text{Number} \mapsto \text{Number} \quad \text{power}.n = 2^n$

$\text{power}.n =$

$n = 0 \rightarrow 1$

$n > 0 \rightarrow 2 * \text{power}.(n-1)$

A traditional recursive  
definition

## A fixed-point definition

$\text{power} = \text{zero} \blacklozenge (\text{minus} \bullet \text{power}) \bullet \text{double}$     the overwriting of disjoint  
where functions is a union

$\text{zero}.n = [0/1]$

$\text{minus}.n = n-1$  for  $n > 0$

$\text{minus}.0 = ?$

$\text{double}.n = 2 * n$

$\text{power} = \text{lim} . ([0/1], [0/1, 1/2], [0/1, 1/2, 2/4], \dots)$



# A CPO of formal languages

$A = \{a_1, \dots, a_n\}$  — an alphabet  
 $\text{Lan}(A) = \{L \mid L \subseteq A^*\}$  — the set of all languages over  $A$   
 $(\text{Lan}(A), \subseteq, \{ \})$  — CPO of formal languages over  $A$

$P, Q : \text{Lan}(A)$

$P \odot Q = \{p \odot q \mid p : P \text{ and } q : Q\}$  — concatenation  
 $P Q = \{p q \mid p : P \text{ and } q : Q\}$  — (an alternative notation)  
 $P^0 = \{\varepsilon\}$   
 $P^n = P P^{(n-1)}$  for  $n > 0$  — n-th power  
 $P^+ = P^1 \mid P^2 \mid \dots$  — plus-power  
 $P^* = P^+ \mid P^0$  — star-power  
 $P^{c^*} = \{(p_1, \dots, p_n) \mid n \geq 0 \text{ and } p_i : P\}$  — Cartesian power

All function defined above, and union, are continuous

Associativity and distributivity

$(P Q) L = P (Q L)$  will be written  $P Q L$

$(P \mid Q) L = (P L) \mid (Q L)$  will be written  $PL \mid QL$

# Equational grammars

polynomial equations

car : Character = {a,...,z,0,...,9}

ide : Identifier = Character | Character © Identifier

exp : Expression = Identifier | {( ) © Expression © {+} © Expression © {}}

In a compact form

car : Character = {a,...,z,0,...,9}

ide : Identifier = Character<sup>+</sup>

exp : Expression = Identifier | ( Expression + Expression )

Equational (polynomial) grammars are equivalent to Chomsky's context-free grammars.

# A CPO of binary relations

$(\text{Rel}(A,A), \subseteq, \{ \})$  — CPO of binary relations  
 $[A] = \{(a, a) \mid a:A\}$  — identity relations (function)  
 $P, R : \text{Rel}(A,A)$   
 $P \bullet R = \{(a, c) \mid (\exists b:B) (a P b \ \& \ b R c)\}$  — composition  
 $R^0 = [A]$   
 $R^n = R \bullet R^{n-1}$  for  $n > 0$   
 $R^+ = R^1 \mid R^2 \mid \dots$   
 $R^* = R^+ \mid R^0$

All function defined above, and union, are continuous

Associativity and distributivity over union

$(P \ R) \ Q = P \ (R \ Q)$  will be written  $P \ R \ Q$   
 $(P \mid R) \ Q = (P \ Q) \mid (R \ Q)$  will be written  $P \ Q \mid R \ Q$

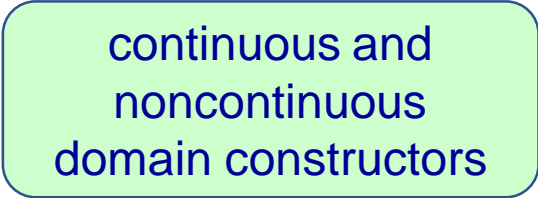
If  $P, R$  - functions, then  $P \bullet R$  - function

# A CPO of domains

(Domain,  $\subseteq$ , { }) — the Cohn's CPO of domains

DEF (M.P. Cohn)

- (1) { }, Identifier, Integer, Character, ... belong to Domain
- (2) Domain is closed under all our domain operations (see below)
- (2) Domain is closed under enumerable unions of sets

$A \mid B$	— set-theoretic union		
$A \cap B$	— set-theoretic intersection		
$A \times B$	— Cartesian product		
$A^{cn}$	— Cartesian n-th power		
$A^{c+}$	— Cartesian plus-iteration		
$A^{c*}$	— Cartesian star-iteration		
FinSub.A	— the set of all finite subsets		
$A \Rightarrow B$	— the set of all mappings including the empty mapping		
$A - B$	— set-theoretic difference		red indicates non-continuity
Sub.A	— the set of all subsets		
$A \rightarrow B$	— the set of all functions from A to B		
$A \mapsto B$	— the set of all total functions from A to B		
Rel.(A,B)	— the set of all relations between A and B		

# Non-continuous domain constructors

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots$  — a chain of mutually different sets  
 $B - A_1 \supseteq B - A_2 \supseteq B - A_3 \supseteq \dots$  — not a chain

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots \rightarrow A$  — a chain of mds  
 $A_1 \rightarrow B \subseteq A_2 \rightarrow B \subseteq A_3 \rightarrow B \subseteq \dots$  — a chain  
 $(\lim \{A_i\}) \rightarrow B \neq \lim \{A_i \rightarrow B\}$

Total functions on  
A not included

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots \rightarrow A$  — a chain of mds  
 $B \rightarrow A_1 \subseteq B \rightarrow A_2 \subseteq B \rightarrow A_3 \subseteq \dots$  — a chain  
 $B \rightarrow (\lim \{A_i\}) \neq \lim \{B \rightarrow A_i\}$

Functions "onto"  
A not included

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \dots \rightarrow A$  — a chain of mds  
 $A_1 \dashv\rightarrow B \subseteq A_2 \dashv\rightarrow B \subseteq A_3 \dashv\rightarrow B \subseteq \dots$  — a chain  
 $(\lim \{A_i\}) \dashv\rightarrow B \neq \lim \{A_i \dashv\rightarrow B\}$

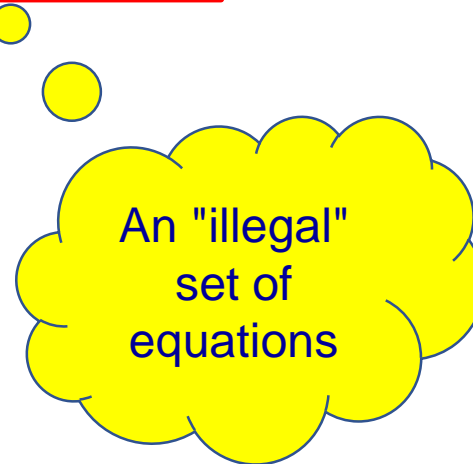
Partial functions  
on A not included

# Domain equations

Data = Number | Record  
Record = Identifier  $\Rightarrow$  Data  
State = Identifier  $\Rightarrow$  Data  
Instruction = State  $\rightarrow$  State

A "legal" set of equations since recursion does not involve the noncontinuous operator  $\rightarrow$ .

State = Identifier  $\Rightarrow$  Data | Procedure  
Procedure = State  $\rightarrow$  State



# Abstract errors — error messages (1/2)

In **Lingua** error messages are generated whenever an operation can't be performed.

E.g. the evaluation of expression  $a/b$  should generate an error whenever:

- variables  $a$  or  $b$  have not been declared at all,
- variables  $a$  or  $b$  have not been declared as numbers,
- the current value of  $b$  is zero,
- the value of  $a/b$  is too large in a given implementation.

Notational convention: The syntax of programs is typeset in `Courier New green`

`dat`: Data — a domain of data  
`dat`: DataE = Data | Error  
`err`: Error — a domain of errors  
(a parameter of our model)

In our model errors are words, e.g.

$a/0$  = 'division-by-zero'

# Abstract errors — error messages (2/2)

$op : Data_1 \times \dots \times Data_n \rightarrow Data$  with computable undefinedness  
 $ope : DataE_1 \times \dots \times DataE_n \mapsto DataE$  coincides with  $ope$  when not error

Transparency for errors:

$ope.(d_1, \dots, d_n) = d_k$  if  $d_k$  the first error in  $(d_1, \dots, d_n)$

Errors may be handled in two ways:

reactively — transparency  
proactively — a restoration mechanism; e.g. in SQL

For technical simplicity  
we assume transparency for most operations in our model



# Propositional calculus of Mc'Carthy

(non-transparent operations)

**if**  $x \neq 0$  **and**  $1/x < 10$  **then**  $x := x+1$  **else**  $x := x-1$  **fi**

If **and** is transparent, then our program aborts for  $x = 0$ .

The solution of John McCarthy:

ff **and-m** ee = ff — lazy evaluation left to right

or-m	tt	ff	ee
tt	tt	tt	tt
ff	tt	ff	ee
ee	ee	ee	ee

and-m	tt	ff	ee
tt	tt	ff	ee
ff	ff	ff	ff
ee	ee	ee	ee

not-m	
tt	ff
ff	tt
ee	ee

# Propositional calculus of Mc'Carthy

## some properties

**and-m, or-m** — associative

$p \text{ and-m } q \neq q \text{ and-m } p$  — not commutative

$p \text{ or-m } (\text{not } p) \neq \text{tt}$  — never false

**and-m** is distributive over **or-m** only on the right-hand side, i.e.

$$p \text{ and-m } (q \text{ or-m } s) = (p \text{ and-m } q) \text{ or-m } (p \text{ and-m } s)$$

# Propositional calculus of Kleene

Even „more lazy” than McCarthy’s calculus

or-k	tt	ff	ee
tt	tt	tt	tt
ff	tt	ff	ee
ee	tt	ee	ee

and-k	tt	ff	ee
tt	tt	ff	ee
ff	ff	ff	ff
ee	ee	ff	ee

not-k	
tt	ff
ff	tt
ee	ee

Now

$p \text{ or-k } q = q \text{ or-k } p$

$p \text{ and-k } q = q \text{ and-k } p$

hence in particular

$p \text{ or-k } ee = ee \text{ or-k } p$

$p \text{ and-k } ee = ee \text{ and-k } p$

If ee may be an infinite computation Kleene's calculus requires a simultaneous evaluation of arguments.





Thank you for  
your attention