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 8th, 2021

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 !: A)	— a is (is not) an element of A
$\{a_1,\ldots,a_n\}$	— 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●g).a = g.(f.a)	— sequential composition of functions
$f = [a_1/b_1,\ldots,a_n/b_n]$	— mapping: $f.a_i = b_i$ and undefined otherwise
[ ]	— empty function
[A].a = a if 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.)

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

$f♦g : A|C \to B|D$ — f overwritten by g

$(f♦g).x =$

$g.x = ! \;➔\; g.x$ if g.x defined then g.x

$g.x = ? \;➔\; f.x$ if g.x undefined then f.x

$f♦g = f \mid g$ — if $f \cap g = \{\}$

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

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

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

…

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

true ➔ f.x in all other cases f.x

MetaSoft notation: sets (domains)

$\{a_1,\ldots,a_n\}$, $\{\ \}$ — a finite set, an empty set
Sub.A, FinSub.A — families of all (all finite) subsets of A,
$A \mid B$ — union of A and B,
A x B — Cartesian product of A and B,
A^{c+} — all finite (nonempty) tuples $(a_1,\ldots,a_n)$; $a_i : A$
( ) — empty tuple
$A^{c*} = A^{c+} \mid \{()\}$ — all finite (possibly 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

Cartesian power

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

typical domain equations:
val : Value = Data x Type
vat : Valuation = Identifier $\Rightarrow$ Value

vat – a metavariable running over Valuation

Chain-complete partially ordered sets

$\sqsubseteq$: Rel(A,A) = {R | R $\subseteq$ A x 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 antisymmetricity

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

$a_1 \sqsubseteq a_2 \sqsubseteq a_3 \sqsubseteq \ldots$ a chain

$\lim(a_i \mid i = 1,2,\ldots)$ limit = least upper bound (if exists)

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

1. every chain in A has a limit,
2. Φ 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 $a_1 \sqsubseteq a_2 \sqsubseteq \ldots$ then $f.a_1 \sqsubseteq f.a_2 \sqsubseteq \ldots$,
2. if $a_1 \sqsubseteq a_2 \sqsubseteq \ldots$ has a limit then $f.a_1 \sqsubseteq f.a_2 \sqsubseteq \ldots$ has a limit,
3. $\lim(f.a_1 \sqsubseteq f.a_2 \sqsubseteq \ldots) = f.[\lim(a_1 \sqsubseteq a_2 \sqsubseteq \ldots)]$.

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 $\lim(f^n.\Phi \mid n = 0,1,2,\ldots)$.

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

Nat = {0,1,2,…} — natural numbers
(Nat → Nat, ⊆, [ ]) — a set-theoretic CPO of partial functions on Nat
f ● g — continuous on both arguments
f ♦ g — continuous on both arguments

power : Number ⟼ Number power.n = 2^n
power.n =
 n = 0 ➔ 1
 n > 0 ➔ 2 * power.(n-1)

A traditional recursive definition

A fixed-point definition

power = zero ♦ (minus ● power) ● double
where
zero.n = [0/1]
minus.n = n-1 for n > 0
minus.0 = ?
double.n = 2 * n

the overwriting of disjoint functions is a union

power = lim.([0/1] , [0/1, 1/2] , [0/1, 1/2, 2/4] , …)

A CPO of formal languages

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

$P, Q : Lan(A)$

$P © Q$	$= \{p © q \mid p : P \text{ and } q : Q\}$	— 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^+	$= P^1 \mid P^2 \mid \ldots$	— plus-power
P^*	$= P^+ \mid P^0$	— star-power
P^{c*}	$= \{(p_1,\ldots,p_n) \mid n \geq 0 \text{ and } p_i : P\}$	— Cartesian power

All function defined above, and union, are <u>continuous</u>

<u>Associativity</u> and <u>distributivity</u>
$(P\ Q)\ L = P\ (Q\ L)$ will be written $P\ Q\ L$
$(P \mid Q)\ L = (P\ L) \mid (Q\ L)$ will be written $PL \mid QL$

Equational grammars (example)

car : Character = {a,…,z,0,…,9}

ide : Identifier = Character | Character © Identifier

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

polynomial equations

In a compact form

car : Character = {`a,…,z,0,…,9`}

`terminal symbols`

ide : Identifier = Character^{+}

exp : Expression = Identifier | `(` Expression `+` Expression `)`

Equational (polynomial) grammars are equivalent to Chomsky's context-free grammars and Backus-Naur grammars

A CPO of binary relations

(Rel(A,A), ⊆, { }) — CPO of binary relations

[A] = {(a, a) | a:A} — identity relations (function)

P, R : Rel(A,A)

P ● R = {(a, c) | (∃b:B) (a P b & b R c)} — composition

R^0 = [A]

R^n = R ● R^{n-1} for n > 0

R^+ = R^1 | R^2 | …

R^* = R^+ | 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 | R) Q = (P Q) | (R Q) will be written P Q | R Q

If P, R – functions, then P ● R – function

A CPO of domains

(Domain, ⊆, { }) — 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 | B	— set-theoretic union
A ∩ B	— set-theoretic intersection
A x 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 ⟹ B	— the set of all mappings including the empty mapping
A – B	— set-theoretic difference
Sub.A	— the set of all subsets
A → B	— the set of all functions from A to B
A ⟼ B	— the set of all total functions from A to B
Rel.(A,B)	— the set of all relations between A and B

continuous and noncontinuous domain constructors

red indicates non-continuity

Non-continuous domain constructors

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

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

Total functions on A not included

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

Functions "onto" A not included

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \ldots \;\boldsymbol{\rightarrow}\; A$ — a chain of mds
$A_1 \longmapsto B \subseteq A_2 \longmapsto B \subseteq A_3 \longmapsto B \subseteq \ldots$ — a chain
$(\lim \{A_i\}) \longmapsto B \quad \neq \quad \lim \{A_i \longmapsto B\}$

Partial functions on A not included

$A_1 \subseteq A_2 \subseteq A_3 \subseteq \ldots \;\boldsymbol{\rightarrow}\; A$ — a chain of mds
$Sub.A_1 \subseteq Sub.A_2 \subseteq Sub.A_3 \subseteq \ldots$ — a chain
$Sub.(\lim \{A_i\}) \quad \neq \quad \lim \{Sub.A_i\}$

Set A not included

Domain equations

$$
\begin{aligned}
&\text{Data} &&= \text{Number} &&| \quad \text{Record} \\
&\text{Record} &&= \text{Identifier} &&\Rightarrow \text{Data} \\
&\text{State} &&= \text{Identifier} &&\Rightarrow \text{Data} \\
&\text{Instruction} &&= \text{State} &&\rightarrow \text{State}
\end{aligned}
$$

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

$$
\boxed{
\begin{aligned}
&\text{State} &&= \text{Identifier} &&\Rightarrow \text{Data} \mid \text{Procedure} \\
&\text{Procedure} &&= \text{State} &&\rightarrow \text{State}
\end{aligned}
}
$$

An "illegal" set of equations

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 : $Data_1$ x…x $Data_n$ $\rightarrow$ Data with computable undefinedness
ope : $DataE_1$ x…x $DataE_n$ $\longmapsto$ DataE coincides with ope when not error

Transparency for errors:

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

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 ≠ 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

error or undefinedness

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 **and-m** q ≠ q **and-m** p — not commutative

p **or-m** (**not** p) ≠ ff — never false

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

p **and-m** (q **or-m** s) = (p **and-m** q) **or-m** (p **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 commutativity

p **or-k** q = q **or-k** p

p **and-k** q = q **and-k** p

hence in particular

tt **or-k** ee = ee **or-k** tt = tt

ff **and-k** ee = ee **and-k** ff = ff

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


Thank you for
your attention