Lingua Project

(12) Metaprograms’ development in Lingua (1)

(Sec. 9.4)

The book "**Denotational Engineering**" may be downloaded from:
https://moznainaczej.com.pl/what-has-been-done/the-book

Andrzej Jacek Blikle

April 26th, 2025

A discipline of validating programming

Formally, validating programming may be regarded as proving theorems of a theory of denotations of **Lingua** — we shall call it **D-theory**.

An example of a specified program — a theorem in **D-theory**

pre x,k **is** integer **and-k** k > 0:	precondition (specification)
x := 0;	
asr x = 0 **rsa**;	assertion (specification)
while x+1 ≤ k **do** x := x+1 **od**	
post x = k	postcondition (specification)

Validating programming consist in deriving:

- correct metaprograms – theorems of **D-theory**
- from other correct metaprograms – theorems of **D-theory**
- by means of program-derivation rules – theorems of **D-theory**

Languages of formalized theories

(illustrated by an example of theoretical arithmetic)

Two syntactic categories:

terms – evaluate to the elements of a universe
formulas – evaluate to truth values tt and ff

2-valued classical logic; classical connectives: **and**, **or**, **not**, **implies**

E.g. in a formalized arithmetic:

Integer	— the universe
x + y	— a free term
2 + 3	— a ground term
x < y	— a free formula
3 < 2	— a ground formula

Semantics of the language of arith.:

$$\text{Validation} = \text{Variable} \longmapsto \text{Integer}$$
$$\text{TerDen} = \text{Validation} \rightarrow \text{Integer}$$
$$\text{ForDen} = \text{Validation} \rightarrow \{\text{tt}, \text{ff}\}$$

valid formula — a formula that is satisfied for all valuations

Examples of valid formulas:

$x < x + 1$

$x < y$ **implies** $x + 1 < y + 1$

$2 < 3+1$

Lingua-V

A language of a (many sorted) formalized theory of denotations (**D-theory**)

many-sorted universe — all domains of denotations in **Lingua** model

Terms (examples)

ide := vex — free term
vex1 < vex2 — free term
x := x+1 — ground term
x < x+1 — ground term

ide, vex, vex1, vex2 run over the corresponding denotations

Formulas (examples)

con1 ⇔ con2 — free formula
con1 ⇔ con2 **implies** con1 ⇒ con2 — free formula (and theorem)
$\sqrt[2]{x} > 2$ ⇔ x > 4 — ground formula (and theorem)

Schemes — free terms and formulas

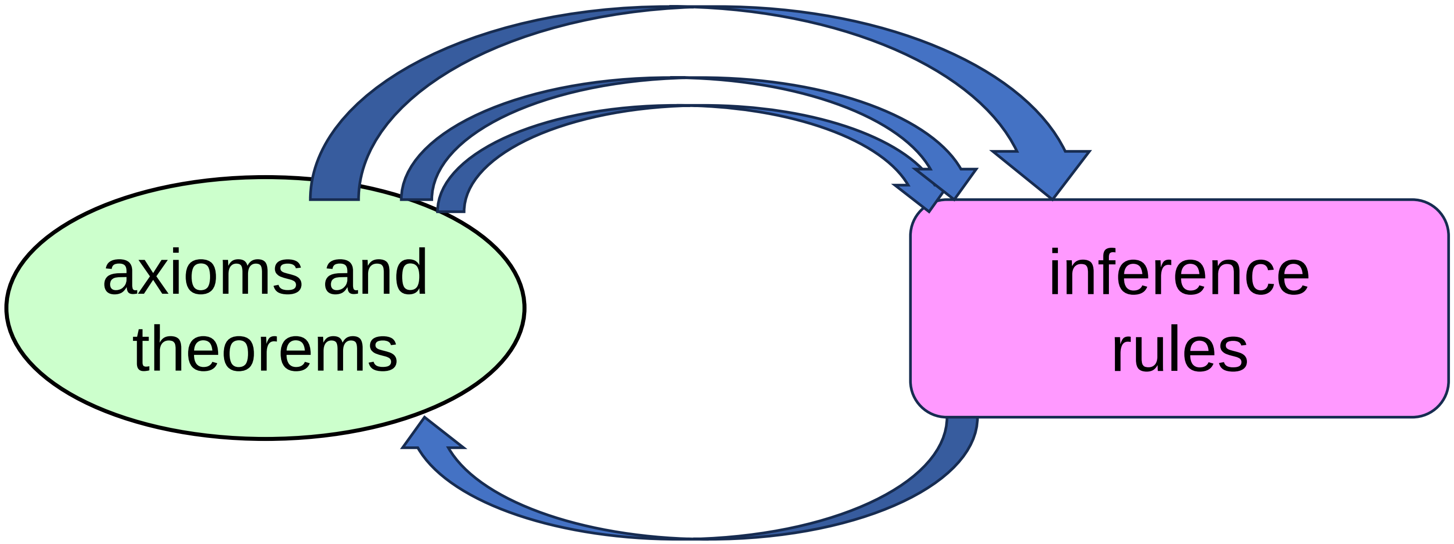
Schemes of instructions:

ide **:=** vex
if vex **then** ins1 **else** ins2 **fi**

A scheme of a metaprogram:

pre con1
  if vex **then** ins1 **else** ins2 **fi**
post vex1 **<** vex2

Proving theorems in formalized theories



Two basic inference rules:

Substitution rule

$$\frac{\models a(x_1,\ldots,x_n)}{\models a(ter_1,\ldots,ter_n)}$$

x_i – variables
ter_i – terms

Detachment rule (modus ponens)

$$\frac{\models for_1,\ \models for_1\ \textbf{implies}\ for_2}{\models for_2}$$

for_i – formulas

Gödel theorem

Every theorem is valid.

$\models$ for – for is a theorem

An ecosystem of programmers in Lingua-V

REPOSITORIES OF BASIC THEOREMS

Repository of general theorems and axioms:

x+y = y+x
x < x+1
con1 ⇔ con2 **implies** con1 ⇒ con2

Repository of schemes of metaprograms:

pre (ide **is free**) **and-k** (tex **is type**)
let ide **be** tex **tel**
post var ide **is** tex

Repository of concrete metaprograms:

pre (length **is free**) **and-k** (real **is type**)
let length **be** real **tel**
post var length **is** real

REPOSITORY OF METAPROGRAMS' CONSTRUCTION RULES

$$\frac{\begin{array}{l}\textbf{pre}\ \text{prd}\ \textbf{:}\ \text{spr}\ \textbf{post}\ \text{poc}\\ \text{poc} \Rightarrow \text{poc1}\end{array}}{\textbf{pre}\ \text{prc}\ \textbf{:}\ \text{spr}\ \textbf{post}\ \text{poc1}}$$

to be read

(**pre** prd : spr **post** poc) **and**
(poc ⇒ poc1)
implies
(**pre** prc **:** spr **post** poc1)

classical

All construction rules in the repository must be proved on the ground of **D-theory**.

Our metaprogram-construction discipline

(the rules of using rules)

Two categories of construction rules

Nuclear rules, e.g.:

pre (ide **is free**) **and-k** (tex **is type**)
  let ide **be** tex **with** yex **tel**
post var ide **is** tex

Implicative rules, e.g.:

$$\frac{\begin{array}{l}\textbf{pre}\ \text{prd} : \text{spr}\ \textbf{post}\ \text{poc} \\ \text{poc} \Rightarrow \text{poc1}\end{array}}{\textbf{pre}\ \text{prc} : \text{spr}\ \textbf{post}\ \text{poc1}}$$

The main metaprograms' derivation rule; **the main rule** (MR):

A metaprogram may be included in a repository of metaprograms exclusively if one of two prerequisites are satisfied:

1. it is created out of a metaprogram in a repository by a replacement of metavariables by compatible terms (substitution), or
2. it is created out of one or more metaprograms from a repository by means of a rule from the repository of rules (detachment).

Conclusion (Gödel):
All (derived) metaprograms in both repositories are correct.

A programmer's view on program development

Each derived metaprogram is eventually of the following general form:

pre prc **:**
  atp-1 **;** … **;** atp-n ; **open procedures ;** asi-1 **;** … **;** asi-k
post poc

were
atp-i's — atomic preambles (declarations or instructions)
asi-i's — atomic instructions (may be structured)

Derivation process:

pre prc-1**:** atp-1 **post** poc-1
…
pre prc-n **: open procedures** **post** poc-n
pre prc-(n+1) **:** asi-1 **post** poc-(n+1)
…
pre prc-(n+k+1) **:** asi-k **post** poc-(n+k+1)

where the metaimplications must be proved:

prc	⇨ prc-1	
poc-i	⇨ prc-(i+1)	for i = 1;k+n
poc-(n+k+1)	⇨ poc	

In this process programmers build:

1. declarations,
2. instructions,
3. conditions

An example of a metaprogram derivation

Mataprogram to be derived

pre x, y **is integer and-k** x > 1**:**
y := x + 1
post y > 0

Rule to be used:

pre prc **:** spr **post** poc
poc ⇨ poc-1

---

pre prc **:** spr **post** poc-1

By substitution we get:

pre pre x, y **is integer and-k** x > 1 : y := x+1 **post** y > 1 — to be found in repository
y > 1 ⇨ y > 0 — to be found in repository

---

pre pre x, y **is integer and-k** x > 1 : y := x+1 **post** y > 0 — to be included in repository

About error-sensitivity

Error-sensitivity of conditions (def):

- error transparent — if [con].sta = error.sta whenever sta carries an error,
- error negative — if [con].sta = fv whenever sta carries an error,
- error positive — if [con].sta = tv whenever sta carries an error,
- error sensitive — if it is error transparent or error negative

Error-sensitive metaprogram (def)

All conditions:

- preconditions,
- assertions
- postconditions

are error sensitive

Metaprogram-construction rules (1)

(general rules)

Lemma 9.4.3-1

If

pre prc **:** spp **; open procedures ;** sin **post** poc

is correct and error-sensitive, then in any execution of the included specprogram that starts with a state satisfying prc:

1. none of spp, sin, poc generates an error,
2. states satisfying prc do not bind identifiers that are declared in spp,
3. all active assertions in sin are satisfied,
4. the terminal state does not carry an error.

Lemma 9.4.4-3 The replacement in a correct metaprograms of its pre- or post-condition or a condition in an assertion by a weakly equivalent condition, does not violate the correctness of the program.

Lemma 9.4.4-4 The replacement in a correct metaprogram of a boolean expression vex in an instruction by a boolean expression vex1 that is strongly equivalent (i.e., vex ≡ vex1) does not violate the correctness of the metaprogram.

Metaprogram-construction rules (2)

Lemma 9.4.4-5 Rule of final composition

pre prc **:** spp **post (**de-con **and-k** sp-con**)**
pre (de-con **and-k** sp-con**)** **: open procedures** **post (**de-con **and-k** op-con **and-k** sp-con**)**
pre (de-con **and-k** op-con **and-k** sp-con**) :** sin **post (**de-con **and-k** op-con **and-k** si-con**)**

---

pre prc**:**
spp **; open procedures ;** sin
post (de-con **and-k** op-con **and-k** si-con**)**

where:

- spp — specified program preamble,
- de-con — hereditary condition induced by declarations included in spp,
- sp-con — condition induced by instructions included in spp,
- op-con — hereditary condition induced by **open procedures**,
- sin — specified instruction,
- si-con — condition induced by sin.

Metaprogram-construction rules (3)

Lemma 9.4.4-6 Rule of sequential composition

pre prc-1**:** spr-1 **post** poc-1
pre prc-2**:** spr-2 **post** poc-2
poc-1 ⇨ prc-2

---

pre prc-1**:** spr-1**;** spr-2 **post** poc-2
pre prc-1**:** spr-1**; asr** poc-1 **rsa;** spr-2 **post** poc-2
pre prc-1**:** spr-1**; asr** prc-2 **rsa;** spr-2 **post** poc-2

Lemma 9.4.4-7 Rule of strengthening precondition

pre prc**:** spr **post** poc
prc-1 ⇨ prc

---

pre prc-1**:** spr**; post** poc

An analogous rule of weakening postcondition is also sound.

Metaprogram-construction rules (3)

Lemma 9.4.4-9 Rule of conjunction and disjunction of conditions

pre prc-1**:** spr **post** poc-1
pre prc-2**:** spr **post** poc-2

---

pre prc-1 **and-k** prc-2 **:** spr **post** poc-1 **and-k** poc-2
pre prc-1 **or-k** prc-2 **:** spr **post** poc-1 **or-k** poc-2

Lemma 9.4.4-10 Rule of propagation of resilient conditions

pre prc**:** spr **post** poc
con **resilient to** spr

---

pre prc **and-k** con **:** spr**; post** poc **and-k** con

Metadeclaration-construction rules (1)

Five categories of atomic declarations to be considered:

1. declarations of value variables,
2. enrichments of covering relations,
3. declaration of classes,
4. global openings of procedures.

Note:
Declarations of

- type constants,
- class attributes,
- methods

formally belong to the cathegory of class transformers.

Lemma 9.4.5-1 Rule of a variable declaration

pre (ide **is free) and-k (**tex **is type)**
  let ide **be** tex **tel**
post var ide **is** tex

Lemma 9.4.5-2 Rule of an enrichment of a covering relation

pre consistent(tex1 **,** tex2**) :** an underivable condition
  enrich-cov(tex1, tex2 **)**
post tex2 **covers** tex1

Metadeclaration-construction rules (2)
(class declarations)

Anchored class transformers

ctd : ClaTraDen = Identifier $\mapsto$ WfState $\rightarrow$ WfState — class transformer denotations

act : AncClaTra = ClaTra **in** Identifier — anchored class transformers

[ctr **in** ide] : WfState $\rightarrow$ WfState

[ctr **in** ide] = [ctr].ide.

a basic scheme
of a class declaration

class
  ide **parent** cli **with**
  ctr-1**;**
  …
  ctr-k
ssalc

an alternative scheme
of a class declaration

class
  ide **parent** cli **with skip-ctr**
ssalc ;
  ctr-1 **in** ide**;**
  …
  ctr-k **in** ide

Metadeclaration-construction rules (2)

(class declarations; cont.)

Lemma 9.4.4-3 Rule of class declaration

pre prc : **class** ide **parent** cli **with skip-ctr ssalc** **post** pa-poc
pre pa-poc : ctr-1 **in** ide **post (**pa-poc **and-k** cr-poc-1**)**
pre (pa-poc **and-k** cr-poc-1**)** : ctr-2 **in** ide **post (**pa-poc **and-k** cr-poc-1**and-k** cr-poc-2**)**
...

---

pre prc**:**
class ide **parent** cli **with** ctr-1**;** … **;** ctr-k **ssalc**
post pa-poc **and-k** cr-poc-1 **and-k** cr-poc-2 **and-k** …

Metadeclaration-construction rules (2)
(the opening of procedures)

pre

pre-proc pr-ide-11 **(val** fpc-v-11 **ref** fpc-r-11**)** body-11 **imperative in** cl-ide-1 **and-k**

pre-proc pr-ide-12 **(val** fpc-v-12 **ref** fpc-r-12**)** body-12 **imperative in** cl-ide-1 **and-k**

…

pre-proc pr-ide-21 **(val** fpc-v-21 **ref** fpc-r-21**)** body-21 **imperative in** cl-ide-2 **and-k**

pre-proc pr-ide-22 **(val** fpc-v-22 **ref** fpc-r-22**)** body-22 **imperative in** cl-ide-2 **and-k**

…

open procedures

post

cl-ide-1.pr-ide-11 **opened and-k**

cl-ide-1.pr-ide-12 **opened and-k**

…

cl-ide-2.pr-ide-21 **opened and-k**

cl-ide-2.pr-ide-22 **opened and-k**

…


Thank you for
your attention