# Lingua Project

## (11) Program correctness in Lingua (2)

(Sec. 9.3)

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

Andrzej Jacek Blikle

April 12th, 2025

# Algorithmic conditions
(repetition)

con : AlgCondition =
  SpePro @ Condition  |  left algorithmic conditions
  Condition @ SpePro  |  right algorithmic conditions

[ ] : AlgCondition ⟼ WfState ⟼ {tv, fv}

[spr @ con].sta =
  (∃ sta1 : {con}) [spr].sta = sta1 ➔ tv
  **true** ➔ fv

i.e. [con].([spr].sta) = tv

[con @ spr].sta =
  (∃ sta1 : {con}) [spr].sta1 = sta ➔ tv
  **true** ➔ fv

We assume that conditions are closed under @.

Since algorithmic conditions are 2-valued, they are unambiguously identified by their truth domains:

{spr @ con} = [spr] ● {con}
{con @ spr} = {con} ● [spr]

None of them is error-transparent and:

- if spr is error transparent and con is error sensitive, then spr @ con is error negative,
- con @ spr need not be error sensitive.

# Error-sensitive conditions

con is error-transparent iff(def) is-error.sta implies [con].sta = error.sta

con is error-negative iff(def) is-error.sta implies [con].sta = fv

con is error-sensitive iff(def) con error-transparent or con error-negative

None of spr@con or con@spr is error-transparent and:

- if spr is error transparent and con is error sensitive, then spr@con is error negative,
- con@spr need not be error sensitive.

A nearly true condition:

[NT].sta =
  is-error.sta ➔ error.sta
  **true** ➔ tv

NT is error-transparent

# A taxonomy of metaconditions

The art of programming in **Lingua** is not reduced to writing declarations and instructions. Equally important is the creation and use of conditions.

Metaconditions describe properties of conditions and programs.

The categories of atomic metaconditions:

1. relational metaconditions represent binary relations between conditions,
2. metaprograms describe properties of specprograms,
3. behavioral metaconditions describe properties of conditions relative to specinstructions or specprograms,
4. temporal metaconditions describe properties of conditions related to their execution-time in correct metaprograms,
5. language-dependent metaconditions describe properties of conditions which are not related to programs, but depend on a programming language where they are used.

Since metaconditions are 2-valued we use classical propositional connectives and quantifiers in compound metaconditions.

# Relational metaconditions
(repetition)

Atomic metaconditions (metapredicates):

| | | | |
|---|---|---|---|
| con1 ⇨ con2 | iff (def) | {con1} ⊆ {con2} | metaimplication; stronger than |
| con1 ⇔ con2 | iff (def) | {con1} = {con2} | weak equivalence |
| con1 ⊑ con2 | iff (def) | [con1] ⊆ [con2] | better definedness; more defined than |
| con1 ≡ con2 | iff (def) | [con1] = [con2] | strong equivalence |

MetaConditions = the least language that includes atomic metaconditions and is closed under 2-valued propositional connectives and quantifiers.

metaconditions are 2-valued

x > 0 **and-kl** $\sqrt[2]{x} > 2$ ≡ x > 4

| | | | |
|---|---|---|---|
| $\sqrt[2]{x} > 2$ | ⇔ | x > 4 | but ≡ does not hold, |
| $\sqrt[2]{x} > 4$ | ⇨ | x > 3 | but neither ⇔ nor ⊑ holds. |
| $\sqrt[2]{x} < 2$ | ⊑ | x < 4 | if $\sqrt[2]{x}$ undefined for x < 0 |

| | | | | |
|---|---|---|---|---|
| con1 | ≡ | con2 | iff | con1 ⊑ con2 and con2 ⊑ con1 |
| con1 | ⇔ | con2 | iff | con1 ⇨ con2 and con2 ⇨ con1 |
| con1 | ≡ | con2 | implies | con1 ⇔ con2 |
| con1 | ≡ | con2 | implies | con1 ⊑ con2 |
| con1 | ⇔ | con2 | implies | con1 ⇨ con2 |

# Relational metaconditions
## (contextual metaconditions)

con1 ≡ con2 **whenever** con means con **and-kl** con1 ≡ con **and-kl** con2
con1 ⇔ con2 **whenever** con means con **and-kl** con1 ⇔ con **and-kl** con2
con1 ⇨ con2 **whenever** con means con **and-kl** con1 ⇨ con **and-kl** con2

context

$n > x^2 \equiv \sqrt[2]{n} > x$ **whenever** ($n \geq 0$ **and-kl** $x \geq 0$)
$n > x^2 \Leftrightarrow \sqrt[2]{n} > x$ **whenever** $x \geq 0$

# Relational metapredicates in the algebra of conditions

**Selected facts:**

(1) ≡ and ⇔ are equivalence relations

(2) ≡ is a congruence wrt **and-kl**, **or-kl** and **not-kl**, e.g.:

if con1 ≡ con2

then (con **and-kl** con1) ≡ (con **and-kl** con2)

(3) ⇔ is a congruence wrt **and-kl** and **or-kl** but not wrt **not-kl**

(4) **and-kl** and **or-kl** are strongly associative

(5) **and-kl** and **or-kl** are strongly commutative

(6) de Morgan laws are strongly satisfied

(7) if [con].sta = ! then [con **or-kl (not** con**)**].sta ≠ fv law of excluded middle

i.e. {con **or-kl (not** con**)**} = {}

(8) if [con].sta = ! then [con **and-kl (not** con**)**].sta ≠ tv law of contradiction

# Three linguistic levels

Lingua – a (classical) programming language
Lingua-V – a language of validating programming metaprograms used to talk about programs
MetaLingua – a language of a 2-valued logic to talk about metaprograms metaconditions are formulas in this logic

**implies-kl** : Condition x Condition ⟼ Condition  constructor in Lingua-V
⇨ : Condition x Condition ⟼ {tt, ff}  constructor in MetaLingua
implies : {tt, ff} x {tt, ff} ⟼ {tt, ff}  classical implication in MetaLingua

(con1 **implies-kl** con2) ≡ **NT** implies con1 ⇨ con2

con1 ⇨ con2 does not imply (con1 **implies-kl** con2) ≡ **NT**.

Despite that the metaimplication $\sqrt[2]{x} > 4 \Rightarrow x > 3$ holds, the condition

$\sqrt[2]{x} > 4$ **implies-kl** $x > 3$

is undefined for $x < 0$.

# Two concepts of program correctness

prc @ spr ⇨ poc  partial correctness of spr relative to prc and poc

prc ⇨ spr @ poc  clean total correctness of spr relative to prc and poc

if poc is error sensitive

prc @ spr is the strongest partial postcondition for spr and prc,

spr @ poc is the weakest total precondition for spr and poc

Correctness of metaprograms

**pre** prc **:** spr **post** poc  iff (def)  prc ⇨ spr @ poc

syntax of metaprograms

The denotations of metaprograms are tt and ff.

# Behavioral metapredicates
## (depend on program's behavior)

| | | |
|---|---|---|
| con **insures LR of** sin | iff | [sin] has limited replicability (LR) in {con} |
| con **resilient to** spr | iff | con **@** spr ⇨ con |
| con **consumed by** spr | iff | con ⇨ spr **@ not-kl** con |
| con **catalyzing for** spr | iff | con ⇨ spr **@** con |
| con **essential for** spr | iff | con ≡ spr **@ NT** con is the weakest precondition for spr |

consumed by and essential for declaration of x

[**NT**].sta = nearly true
- is-error.sta ➔ error.sta
- **true** ➔ tv

resilient

```
pre (x is free) and-kl (var y is integer) :
   let x be real tel;
   asr val x is real rsa
   x := 17,3
   …
post (var x is real) and-kl (var y is integer) and-kl (x = 17,3)
```

catalyzing and essential

# Temporal metapredicates
## (execution-time dependent)

A cut of a metaprogram

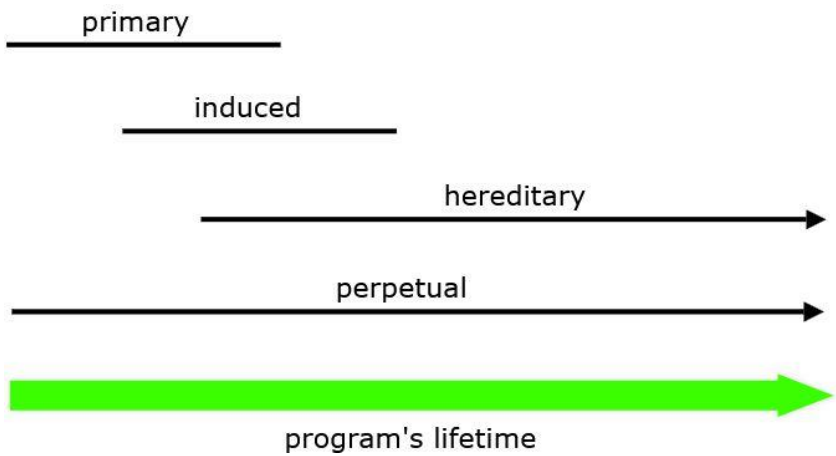
mpr = **pre** prc **:** spr **post** poc

is a pair (head, tail) such that spr = head **;** tail

and **;** is not in the body of a procedure

Let mpr = **pre** prc**:** spr **post** poc

| | | |
|---|---|---|
| con **primary in** mpr | iff | con satisfied at the entrance |
| con **induced in** mpr | iff | con is satisfied in a cut |
| con **hereditary in** mpr | iff | con once satisfied is satisfied in all later cuts |
| con **co-hereditary in** mpr | iff | con once falsified is false in all later cuts |
| con **perpetual in** mpr | iff | con is primary and hereditary |



# Temporal metapredicates

Let mpr = **pre** prc**:** spr **post** poc

| | | |
|---|---|---|
| con **primary in** mpr | iff | con satisfied at the entrance |
| con **induced in** mpr | iff | con is satisfied in a cut |
| con **hereditary in** mpr | iff | con once satisfied is satisfied in all later cuts |
| con **co-hereditary in** mpr | iff | con once falsified is false in all later cuts |
| con **perpetual in** mpr | iff | con is primary and hereditary |

```
pre (x is free) and-kl (var y is integer) :
   let x be real tel;
   asr val x is real rsa
   x := 17,3
   …
post (var x is real) and-kl (var y is integer) and-kl (x = 17,3)
```

| | |
|---|---|
| x **is free** | — is primary and co-hereditary, |
| **var** y **is** integer | — is perpetual |
| **var** x **is** real | — is induced and hereditary, |
| x = 17,3 | — is induced but not necessarily hereditary. |

# Language related metapredicates
(program independent; universal)

con **is immunizing** iff hereditary in every program
e.g., **var** ide **is real**

con **is immanent** iff never false; may be tt, ?, ee
e.g., x + y = y + x

con **is underivable** iff must be assumed in prc to be satisfied
e.g., x **is free**

**Two general rules of handling conditions:**

- if we need an underivable condition in a program, we have to put it to precondition,
- whenever a hereditary condition appears somewhere in the program, we can add it to the postcondition; strongest postcondition is a conjunction of all hereditary conditions


Thank you for
your attention