Lingua Project

(4) Usability and visibility of items

(Sec. 5)

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

Andrzej Jacek Blikle

February 15th, 2025

A recapitulation of former lecture

Classes

cla	: Class	= Identifier x TypEnv x MetEnv x Objecton	classes
tye	: TypEnv	= Identifier ⟹ Type | {Θ}	type environments
mee	: MetEnv	= Identifier ⟹ Method	method environments
met	: Methods	= ProSig | PrePro	methods

A recapitulation of former lecture

Stores and states

sta : State = Env x Store — states
env : Env = ClaEnv x ProEnv x CovRel — environments
cle : ClaEnv = Identifier ⟹ Class — class environments
pre : ProEnv = ProInd ⟹ Procedure — procedure environments
pri : ProInd = Identifier x Identifier — procedure indicators
sto : Store = Objecton x Deposit x OriTag x SetFreTok x (Error | {'OK'}) — stores
cov : CovRel = Sub.((DatTyp x DatTyp) | (ObjTyp x ObjTyp)) — covering relations
sft : SetFreTok = Set.Token — sets of (free) tokens

Auxiliary function

get-tok : SetFreTok ⟼ Token x SetFreTok
get-tok.sft = (tok, sft − {tok})) such that tok : sft

{$} | Identifier

wskaźnik pochodzenia

An objecton my-obn is said to be well-formed in a state
sta = ((cle, pre, cov), (obn, dep, ota, sft, err)), if:
(1) for any attribute ide, if obn.ide = !, and dep.(obn.ide) = !, then:
obn.ide **VRA.cov** dep.(obn.ide) — value by reference acceptability (see later),
(2) all inner objectons of obn are well-formed in sta.

A class (ide, tye, mee, obn) is said to be well-formed in a state, if
(1) obn is well-formed in this state,
(2) for every reference (tok, (typ, yok, ota)) in obn, its origin tag ota is either $ or ide

A

A recapitulation of former lecture

Well formed states

A state sta = ((cle, pre, cov), (obn, dep, ota, sft, err)) said to be well-formed, if:

1. obn is well formed in sta,
2. external names of all classes declared in cle coincide with their internal names,
3. all surface and inner objects in obn are of types that are the names of classes declared in cle,
4. all classes declared in cle are well-formed,
5. sft includes only such tokens that do not appear in references bound in dep,
6. every identifier appearing in a state, appears in it only once; e.g., if an identifier is a variable, it can't be at the same time a type constant or a class name.

WfState − the set of all well-formed states

Auxiliary functions:

error : Store ⟼ Error | {'OK'} error : State ⟼ Error | {'OK'}
is-error : Store ⟼ Boolean is-error : State ⟼ Boolean

(env, (obn, dep, ota, sft, err)) ◀ new-err = (env, (obn, dep, sft, ota, new-err))
(env, (obn, dep, ota, sft, err)) ◀ new-sft = (env, (obn, dep, new-sft, ota, err))

declared : Identifier x State ⟼ {tt, ff}

A

Two regimes of handling items

An informal overview

The usability regime defines restrictions about the use of values depending on their types and the yokes of references:

- when they are sent to value constructors (yokes not involved),
- when they are assigned to references (yokes involved),
- when they are sent as actual parameters to procedure calls (yokes not involved).

The visibility regime defines restrictions about the use of values depending on a programming context:

- procedure-dependent visibility: all items locally declared in procedure bodies will be visible exclusively in these bodies,
- class-dependent visibility: selected items in classes may be declared as private.

Covering relations between types

Usability regime

- The types of values assigned to references must be acceptable by the types of references, and the values themselves must satisfy the yokes of references.
- The types „expected” by value constructors must accept the types of their arguments.

cov : CovRel = Sub.((DatTyp x DatTyp) | (ObjTyp x ObjTyp))

E.g. (’integer’, ’small integer’) : cov
(’employee’, ’accountant’) : cov

cov = Ld-cov | Pr-cov
Ld- language designer
Pr- programmer

TTA.cov ⊆ Type x Type

type-by-type acceptability relation
reflexivity + transitivity

VRA.cov ⊆ Reference x Value

value-by-reference acceptability relation
TTA.cov + yokes in references

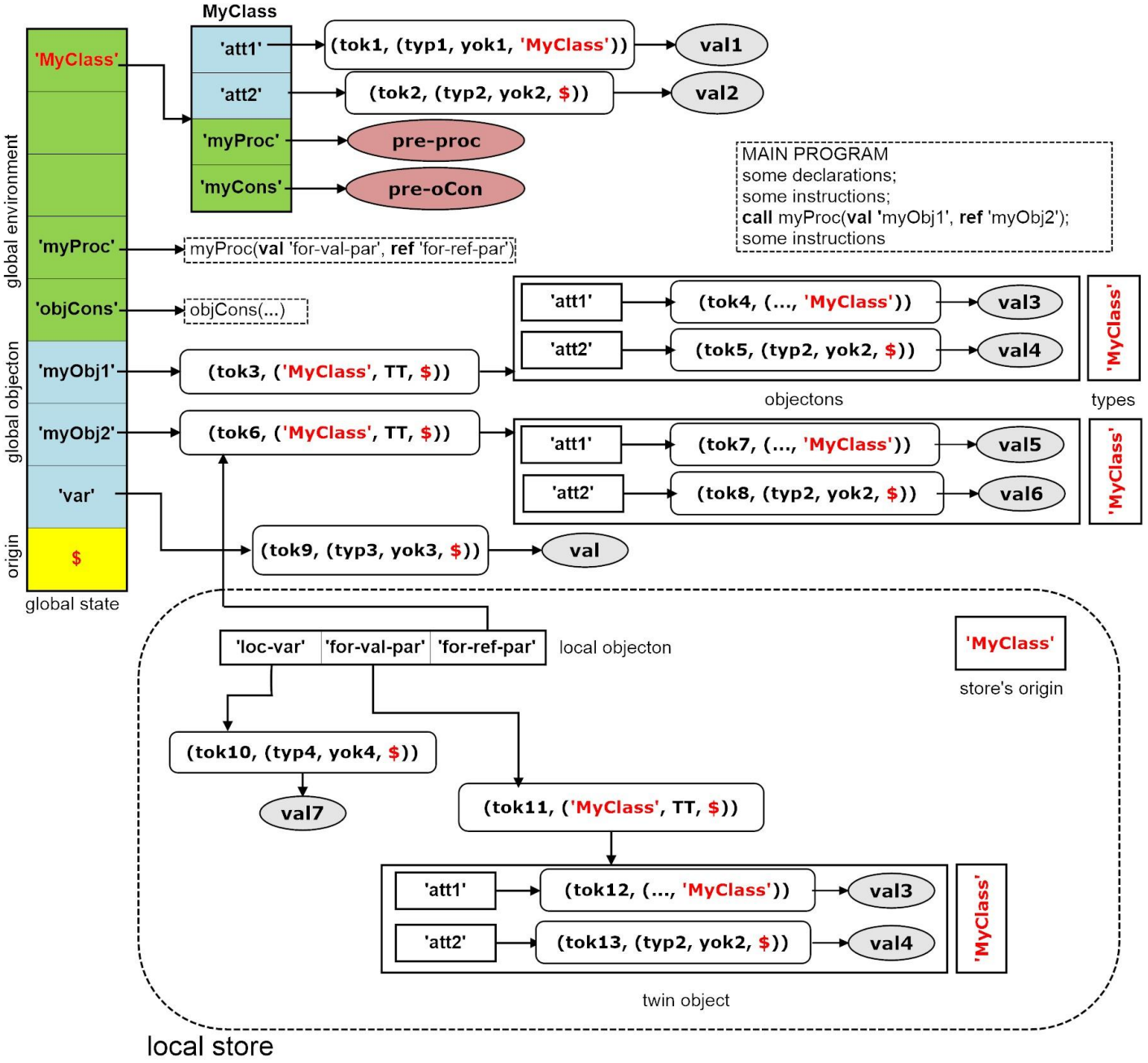
Definitions in Sec. 5.4.2

A

Visibility of references
Basic rules

1. A reference is visible in a state, if the origin tag of this reference
   1. either is $ (global visibility), or
   2. coincides with the origin tag of the state (local visibility).
2. A reference must be visible whenever we intend to:
   1. get a value assigned to it in evaluating an expression,
   2. change the value assigned to it in executing an assignment instruction.
3. The origin tags of references and states are established when these references and states are created, and later they can't be changed.
4. Variables declared in states are always public.
5. Attributes declared in MyClass may be
   1. public; reference origin tag is $,
   2. private; reference origin tag is MyClass
6. Local states of procedures in MyClass have origin tag MyClass
7. All procedures are global

A


MyClass
'att1'
'att2'
'myProc'
'myCons'
(tok1, (typ1, yok1, 'MyClass'))
(tok2, (typ2, yok2, $))
pre-proc
pre-oCon
val1
val2
MAIN PROGRAM
some declarations;
some instructions;
call myProc(val 'myObj1', ref 'myObj2');
some instructions
global environment
'MyClass'
'myProc'
'objCons'
'myObj1'
'myObj2'
'var'
$
origin
global state
global objecton
myProc(val 'for-val-par', ref 'for-ref-par');
objCons(...)
(tok3, ('MyClass', TT, $))
(tok6, ('MyClass', TT, $))
(tok9, (typ3, yok3, $))
val
'att1'
'att2'
(tok4, (..., 'MyClass'))
(tok5, (typ2, yok2, $))
val3
val4
'MyClass'
objectons
types
'att1'
'att2'
(tok7, (..., 'MyClass'))
(tok8, (typ2, yok2, $))
val5
val6
'MyClass'
'loc-var'
'for-val-par'
'for-ref-par'
local objecton
'MyClass'
store's origin
(tok10, (typ4, yok4, $))
val7
(tok11, ('MyClass', TT, $))
'att1'
'att2'
(tok12, (..., 'MyClass'))
(tok13, (typ2, yok2, $))
val3
val4
'MyClass'
twin object
local store


Thank you for
your attention