Lingua Project
(8) Declarations

(Sec. 6.7)

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

Andrzej Jacek Blikle

March 22nd, 2025

An overview of declarations

dcd : DecDen = WfState → WfState

1. at the **level of states**
   a. they assign references, classes and procedures to identifiers,
   b. they assign values to references in deposits.
   c. they modify covering relations (declarations do not modify anything else!)
2. at the **level of classes**
   a. they assign references, pre-procedures and types to identifiers,
   b. they assign values to references in deposits.

Constructors of declarations

var-dec	: ListOfIde x TypExpDen	⟼ DecDen	variable declaration
enrich-cov-rel	: TypExpDen x TypExpDen	⟼ DecDen	enrichments of cov. rel.
cla-dec	: Identifier x ClaInd x ClaTraDen	⟼ DecDen	class declaration
pro-open	:	⟼ DecDen	procedure opening
skip-dec	:	⟼ DecDen	trivial declaration
compose-dec	: DecDen x DecDen	⟼ DecDen	composition of decl.

cli : ClaInd = {‘empty-class’} | Identifier class indicator

Declarations of variables

var-dec : ListOfIde x TypExpDen x YokExpDen ⟼ DecDen i.e.
var-dec : ListOfIde x TypExpDen x YokExpDen ⟼ WfState → WfState
var-dec.(loi, ted, yed).sta =
  is-error.sta ➔ sta
  loi = () ➔ sta ◄ 'empty list of variables can't be declared'
  let
    ((cle, pre, cov), (obn, dep, ota, sft, 'OK')) = sta
    (ide-1,…,ide-n) = loi
  ted.sta : Error ➔ sta ◄ ted.sta
  declared.ide-i.sta ➔ sta ◄ 'identifier declared' for i = 1;n
  yed.sta = ? ➔ ?
  let
    de-typ = ted.sta declared type
    de-yok = yed.sta declared yoke
    (tok-1, sft-1) = get-tok.sft
    (tok-i, sft-i)) = get-tok.sft(i-1) for i = 2;n
    ref-i = (tok-i, (de-typ, de-yok, $))
  de-typ : ObjTyp **and** cle.de-typ = ? ➔ 'class unknown'
  true ➔ ((cle, pre, cov), (obn[ide-i/ref-i | i = 1;n], dep, ota, sft-n, 'OK'))

An overview of class declarations

The declarations of classes in three steps:

1. the choice of a parent class (empty or previously declared),
2. the removal of all procedure- and type- declarations (an engineering decision),
3. the enrichment of the class by new items (done by class transformers):
   a. attributes,
   b. type constants,
   c. pre-procedures.

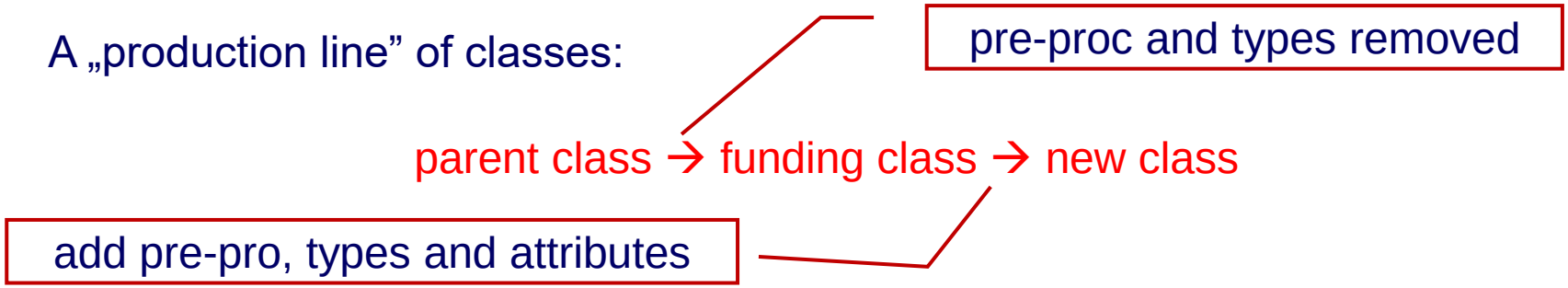
Will be public

```
class MyClass:
    par HisClass by:                                  parent class
        let age be integer as private tel;            attribute dec.
        set worker as HisClass.employee tes           type constant dec.
        proc promote(val cfp-v ref cfp-r) prc corp    procedure dec.
ssalc
```

Making a funding class

Funding class is a source of the (future) declared class to be enriched by means of class transformers.

A „production line” of classes:

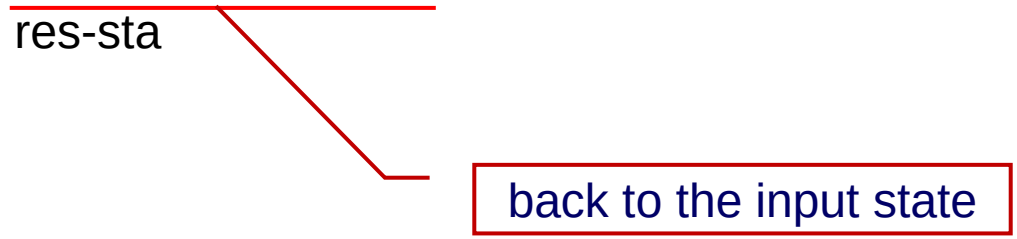


cli : ClaInd = {‘empty-class’} | Identifier class indicators

make-funding-class : ClaInd ⟼ Identifier ⟼ WfState ⟼ Class | Error
make-funding-class.cli.ide.sta =
  cli = ‘empty-class’ ➔ (ide, [ ], [ ], [ ])
  let
    ((cle, pre, cov), sto) = sta
  cle.cli = ? ➔ ‘parent class unknown’
  let
    (cl-ide, tye, mee, obn) = cla.cli parent class
    [ide-1/ppr-1,…,ide-n/ppr-n, ide-(n+1)/sig-1,…,ide-(n+k)/sig-k]= mee
  true ➔ (ide, [ ], [ide-(n+1)/sig-1,…,ide-(n+k)/sig-k], obn)

Class declarations

```
cla-dec : Identifier x ClaInd x ClaTraDen ⟼ DecDen
cla-dec : Identifier x ClaInd x ClaTraDen ⟼ WfState ⟼ WfState
cla-dec .(de-ide, pa-cli, ctd).sta =                de- for "declared"; pa- for "parent"
  is-error.sta              ➔ sta
  declared.de-ide.sta       ➔ sta ◄ 'identifier already declared'
  let
    ((cle, pre, cov), sto)   = sta
    fu-cla                   = make-funding-class.pa-cli.de-ide.sta    funding class
  fu-cla : Error            ➔ sta ◄ fu-cla
  let
    fu-sta = ((cle[de-ide/fu-cla], pre, cov), sto)                     funding state
  ctd.de-ide.fu-sta = ?     ➔ ?                    ctd class-transformation denotation
  let
    res-sta= ctd.de-ide.fu-sta      resulting state (with an enriched funding class)
  is-error.res-sta          ➔ sta ◄ error.res-sta
  true                      ➔ res-sta
```



Class transformers (1)

ctd : ClaTraDen = Identifier ⟼ WfState → WfState

Class transformers add to classes:

- attributes,
- types,
- methods.

```
add-abs-att          : Identifier x TypExpDen x PriSta                ⟼ ClaTraDen
concretize-abs-att   : Identifier x ValExpDen                         ⟼ ClaTraDen
add-con-att          : Identifier x ValExpDen x TypExpDen x PriSta    ⟼ ClaTraDen

add-abs-typ          : Identifier                                     ⟼ ClaTraDen
concretize-typ       : Identifier x TypExpDen                         ⟼ ClaTraDen
add-con-typ          : Identifier x TypExpDen                         ⟼ ClaTraDen

add-abs-imp-met      : Identifier x ImpProSigDen                      ⟼ ClaTraDen
concretize-imp-met   : Identifier x ImpProSigDen x ProDen             ⟼ ClaTraDen
add-con-imp-met      : Identifier x ImpProSigDen x ProDen             ⟼ ClaTraDen

add-abs-fun-met      : Identifier x FunProSigDen                      ⟼ ClaTraDen
concretize-fun-met   : Identifier x FunProSigDen x ProDen x ValExpDen ⟼ ClaTraDen
add-con-fun-met      : Identifier x FunProSigDen x ProDen x ValExpDen ⟼ ClaTraDen
```

Class transformers (2)

add-abs-obj-met	: Identifier x ObjConSigDen	⟼ ClaTraDen
concretize-obj-met	: Identifier x ObjConSigDen x ProDen	⟼ ClaTraDen
add-con-obj-met	: Identifier x ObjConSigDen x ProDen	⟼ ClaTraDen
compose-cla-tra	: ClaTraDen x ClaTraDen	⟼ ClaTraDen

Adding an abstract attribute to a class

add-abs-att : Identifier x TypExpDen x YokExpDen x PriSta ⟼ Identifier ⟼
WfState → WfState

add-abs-att.(at-ide, ted, yed, pst).cl-ide.sta =
  is-error.sta ➔ sta
  declared.at-ide.sta ➔ sta ◄ 'attribute already declared'
  ted.sta : Error ➔ sta ◄ ted.sta
  yed.sta = ? ➔ ?
  let
    ((cle, pre, cov), (obn, dep, ota, sft, 'OK')) = sta
    yok = yed.sta
  cle.cl-ide ➔ 'class unknown'
  let
    (cl-ide, tye, mee, obn) = cle.cl-ide
    de-typ = ted.sta
    (tok, new-sft) = get-tok.sft
    ref =
      pst = 'public' ➔ (tok, (de-typ, yok, $))
      pst = 'private' ➔ (tok, (de-typ, yok, cl-ide))
    new-cla = (cl-ide, tye, mee, obn[at-ide/ref])
  true ➔ ((cle[cl-ide/new-cla], pre, cov), (obn, dep, ota, new-sft, 'OK'))

de-typ – declared type

privacy status

Adding an imperative pre-procedure to a class

add-con-imp-met : Identifier x ImpProSigDen x ProDen ⟼ ClaTraDen i.e.
add-con-imp-met : Identifier x ForParDen x ForParDen x ProDen ⟼
⟼ Identifier → WfState → WfState
add-con-imp-met.(pr-ide, fpd-v, fpd-r, prd).cl-ide.sta =
  is-error.sta ➔ sta
  let
    ((cle, pre, cov), sto) = sta
    (v-ide-1,…, v-ide-n) = get-parameters.fpd-v
    (r-ide-1,…, r-ide-k) = get-parameters.fpd-r
  cle.cl-ide = ? ➔ sta ◄ 'class unknown'
  declared.pr-ide.sta ➔ sta ◄ 'identifier not free'
declared.v-ide-i.sta ➔ sta ◄ 'identifier not free' for i = 1;n
declared.r-ide-i.sta ➔ sta ◄ 'identifier not free' for i = 1;k
  let
    (cl-ide, tye, mee, obn) = cle.cl-ide
  let
    ipp = create-imp-pre-pro.(fpd-v, fpd-r, prd, cl-ide)
    new-cla = (ide, tye, mee[pr-ide/ipp], obn)
  true ➔ ((cle[cl-ide/new-cla], pre, cov), sto)

Enriching type-covering relation

enrich-cov-rel : TypExpDen x TypExpDen ⟼ DecDen i.e.
enrich-cov-rel : TypExpDen x TypExpDen ⟼ WfState ⟼ WfState
enrich-cov-rel.(ted-1, ted-2).sta =
  is-error.sta ➔ sta
  ted-i.sta : Error ➔ sta ◄ ted-i for i = 1,2
  let
    typ-i = ted-i.sta for i = 1,2
    ((cle, pre, cov), (obn, dep, ota, sft, 'OK')) = sta
    cov-1 = enrich-cov.(cov, typ-1, typ-2)
  cov-1 : Error ➔ sta ◄ cov-1
  typ-1, typ-2 /: ObjTyp ➔ ((cle, pre, cov-1), (obn, dep, ota, sft, 'OK'))
  true ➔
    let
      ide-i = typ-i for i = 1,2
    cle.ide-i = ? ➔ 'object types must point to declared classes' for i = 1,2
    true ➔ ((cle, pre, cov-1), (obn, dep, ota, sft, 'OK'))

Opening procedures
(a half formal definition)

get-pre-pro : WfState ⟼ (ProInd x PrePro)c* get pre-procedures

create-open-pro-den : ⟼ ProOpeDen
create-open-pro-den : ⟼ WfState ⟼ WfState

create-open-pro-den .().dt-sta = dt-sta declaration-time state
  is-error.dt-sta ➔ dt-sta
  get-pre-pro.dt-sta = () ➔ dt-sta ◄ 'no procedures to declare'
  let
    ((pri-1, prp-1),…,(pri-n, prp-n)) = get-pre-pro.dt-sta
    ((dt-cle, dt-pre, dt-cov), dt-sto) = dt-sta
    pro-1 = prp-1.(dt-cle, dt-pre[pri-1/pro-1,…, (pri-n/pro-n], dt-cov)
    …
    pro-n = prp-n.(dt-cle, dt-pre[pri-1/pro-1,…, (pri-n/pro-n], dt-cov)
    ot-env = (dt-cle, dt-pre[pri-1/pro-1,…, (pri-n/pro-n]), dt-cov) open-time env.
true ➔ (ot-env, dt-sto)


Thank you for
your attention