Types avancés

Chapitre 9

Informations associées à un type

Son nom

Une information abstraite (*domaine abstrait*)

- Ensemble de valeurs (abstraites)
- Ensemble d'opérations (abstraites sur les valeurs abstraites)

Une information physique (*domaine concret*)

- Une représentation des valeurs en machine
- Une représentation des opérations en machine (algorithme codé)

Typage et implantation : exemple

Le type entier (son **nom** `Integer`)

Domaine abstrait : l'ensemble infini des entiers muni des opérations arithmétiques (+,-,/,*,…)

Domaine concret : représentation binaire des entiers représentables par des mots de 32 bits $[-2^{31}, +2^{31}-1]$ et représentation des opérations au moyen des opérations binaires de la machine

Contraintes physiques

Tous les entiers ne sont pas représentables

Perte des propriétés de l'ensemble abstrait des valeurs

$2^{16}*2^{16}=2^{32}$ => non représentable

Perte de l'associativité : supposons que le plus grand entier représentable soit 1000

si `600+500+(-400)` est évalué ainsi : `600+(500-400)` alors le calcul est correct, alors que l'évaluation `(600+500)-400` lève une erreur car 1100 n'est pas représentable.

On dit qu'il y a *débordement*.

Notion de sous-type

Un sous-type définit un sous-ensemble des valeurs d'un type appelé type de base

Les opérations sur les valeurs du sous-type sont les opérations sur les valeurs du type de base

Déclaration d'un sous-type

```
<déclaration_sous_type>::=
    subtype <ident_de_sous_type> is
        <type_de_base> range <contrainte>;
```

Exemples

```
subtype NB_JOURS is Integer range 1..31;
subtype NB_JOURS_MOIS is NB_JOURS range 28..31;
subtype JOURS_MOIS is NB_JOURS;
X:Integer range 50..200;
-- sous-type anonyme
```

sous-types : remarques

Les bornes des intervalles ne sont pas nécessairement statiques.

L'ensemble des valeurs d'un sous-type n'est donc pas obligatoirement connu à la compilation.

```
declare
  N:Integer;
begin
  get(N);
  declare
    subtype Intervalle is Integer range 0..N;
  begin
    …
  end;
end;
```

Limite de la notion de type

```
V1:Float;
V2:Float:= 221.2;
T :Float:= 64.5;
V1:=V2*T;
-- opération licite, mais
-- aucune vérification sémantique
-- sur les valeurs
```

Types dérivés

Déclaration

<déclaration_type_dérivé>::=
type *<ident_type_dérivé>* ***is***
new *<définition_type_dérivé>;*

Exemples

```
type Vitesse is new Float;
type Temps is new Float;
V1:Vitesse;V2:Vitesse:=221.2;
T:Temps:=64.5;
V1:=V1*T;
-- opération illicite, erreur de typage
```

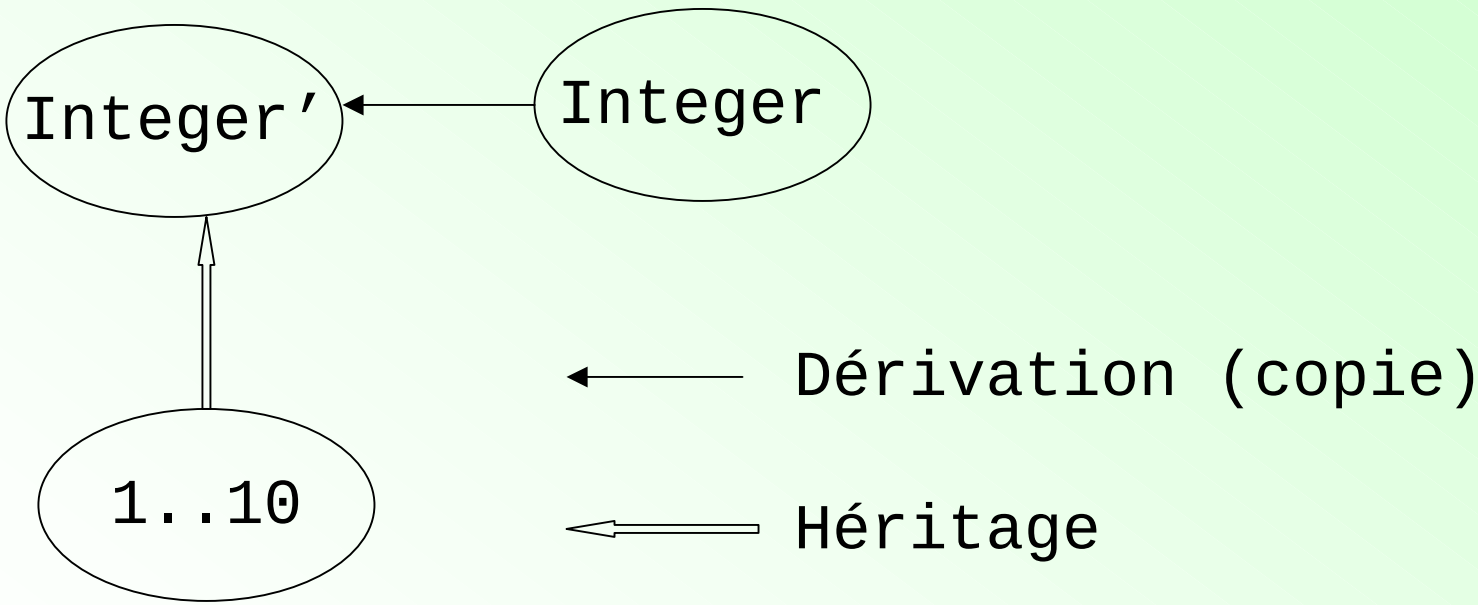
Définition d'un type dérivé

Un *type dérivé* définit un nouveau type qui possède le même domaine abstrait et le même domaine concret que le type dont il dérive, appelé type père.

- L'ensemble des valeurs du type dérivé est une **copie** de l'ensemble des valeurs du type père
- Les opérations sur les valeurs du type dérivé sont identiques aux opérations du type père

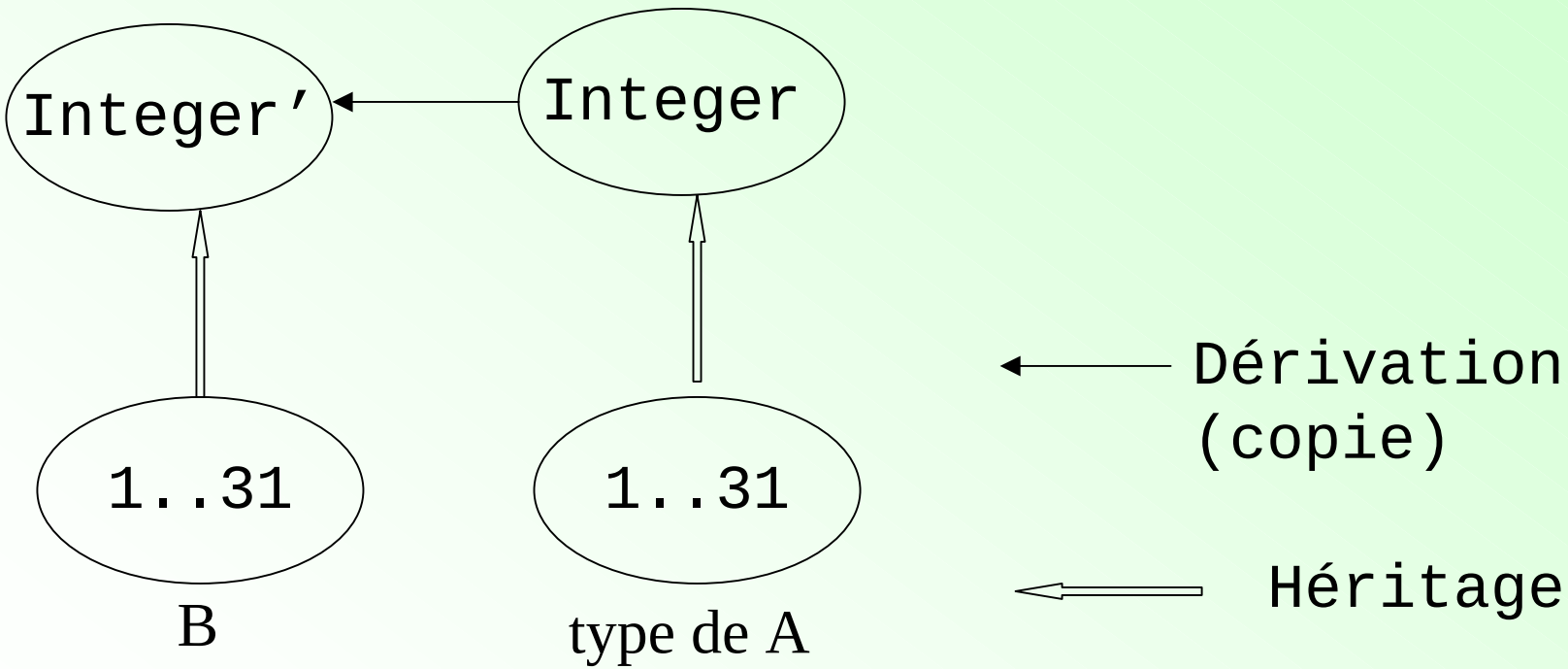
Construction d'un type dérivé (1)

```
type A is new Integer range 1..10;
```



Construction d'un type dérivé (2)

```
A:Integer range 1..31;--sous-type anonyme
type B is new Integer range 1..31;
```



Compatibilité entre types

```
declare
   X:Integer range 1..100:=5;
   -- sous-type anonyme
   subtype Jour is Integer range 1..31;
   -- sous-type
  type Semaine is new Integer range 1..7;
  -- type dérivé
   Y:Jour;
   Z:Semaine;
begin
  Y:=X;
-- affectation licite
  Z:=X;
-- affectation illicite
  Y:=53;
-- CONSTRAINT_ERROR
end;
```

Conversion de types (1)

Transformation d'une valeur V d'un type en une valeur V' d'un autre type :

Cas n°1 : le domaine concret de V est inclus dans celui de V'

$[1..10] \subseteq N$

Cas n°2 : le domaine abstrait de V est inclus dans celui de V'

$N \subseteq R$

Cas n°3 : le domaine abstrait de V contient celui de V'

$R \subseteq N$

Conversion de types (2)

Cas n°1

les représentations physiques des valeurs de `[1..10]` et celles de `[-2^31,+2^31-1]` sont identiques

Cas n°2

il faut transformer la représentation des valeurs de `N` en celles de `R`

Cas n°3

pour transformer la représentation des valeurs de `R` en celles de `N`, il faut passer par une approximation

Conversions implicites

Types entiers vers types entiers

Types réels vers types réels

Conversion explicites

Elles sont réalisées par la construction :

<conversion explicite>::=
*<type_cible>**(**<expression_source>**)***

Exemples

- Entre numériques
  `Float(2*J)  Integer(3.14)`
- Entre types dérivés
  ```
  type Point is new Complexe;
  X:Complexe:=(5.3,9.8);
  P:Point;
  P:=Point(X);X:=Complexe(P);
  ```

Conversion entre tableaux

Pour convertir un tableau dans un autre tableau, il faut qu'ils aient :

- Même dimension
- Même nombre d'indices
- Même type d'indice
- Même type de composant

Exemples

```
type T1 is array(1..5) of Integer;
type T2 is array(4..9) of Integer;
X:T1:=(5,4,7,9,1);Y:T2:=(2,6,7,0,4);
 X:=T1(Y);Y:=T2(X);
```

Type article paramétré

Permettre de définir des types article dont la structure est fonction de paramètres (appelés *discriminants*).

Certains champs du type deviennent variants. Ils dépendent de la valeur du paramètre.

Les paramètres du type (*discriminants*) sont des champs particuliers du type.

Une dépendance est ainsi établie entre les *discriminants* et certains autres champs

Déclaration d'un type article paramétré

Syntaxe

```
<déclaration_type_article_paramétré>::=
 type <ident_type>
  (<discriminant>:<type_discret>[:=<exp>]
   {;<discriminant >:<type_discret>[:=<exp>]})
 is
   record
     <champ>:<type>[:=<exp>];
    {<champ>:<type>[:=<exp>];}
   end record;
```

Type article paramétré : exemple

```
type Employe (BORNE:Positive:=10) is
  record
    nom:String(1..BORNE);
    naissance:Date;
    salaire:Float:=SMIC;
  end record;
 directeur:Employe(BORNE=>6);
 directeur.nom:=“Patron”;
 marcel:Employe;
```

Type article paramétré : exemple

```
type Paire(BSUP:Integer:=100) is
 record
  gauche,droite:Integer range 1..N;
 end record;

-- accès au discriminant
P:Paire;
Ada.Integer_Text_io.put(P.BSUP);
-- sous typage de Paire, affiche 100
Q:Paire(50);
-- sous-type anonyme
subtype Paire_50 is Paire(BSUP=>50);
Q:Paire_50;
Q:=P;
-- affectation illégale
```

Type article paramétré

Modification de la valeur d'un discriminant

```
P:Paire;
P.BSUP:=80;
-- illégal
```

Modification globale possible

```
P:Paire:=(80,4,25);
```

Type article avec partie variante

Il est possible de définir des types dans lesquels une partie de la structure est fixe et l'autre variante.

Le choix de la partie variante, qui détermine la structure effective des valeurs du type, dépend de la valeur d'un discriminant.

Ces types reflètent la somme disjointe d'une famille d'ensembles de valeurs.

Type article avec partie variante : syntaxe

```
<déclaration_type_variant>::=
type<identificateur_de_type>
          (<etiquette>:<type_discret>)is
record
  case
   when <choix>{ | <choix>} =><liste_de_champs>;
  {when <choix>{ | <choix>} =><liste_de_champs>;}
  end case;
end record;
```

Exemple (1/2)

```
type Genre is(masculin,feminin);
type Personne(sexe:Genre) is
  record
    naissance:Date;
    case sexe is
      when masculin=>barbe:Boolean;
      when feminin=> enfants:Natural;
    end case;
  end record;
```

Exemple (2/2)

Tous les objets du type sont contraints car aucune valeur par défaut n'est donnée au discriminant

```
paul:Personne(masculin);
jeanne:Personne(feminin);
```

ou

```
subtype Homme is Personne(sexe=>masculin);
pierre:Homme;
```

Affectation globale

```
pierre:=(masculin,(12,2,1950),false);
```

Accès aux composants

```
pierre.barbe:=true;
```

Modélisation de données : les cartes à jouer (1/2)

```
type Couleur is (trefle,carreau,coeur,pique);
type Figure is(as,roi,dame,valet,petite);
type Niveau(F:Figure) is
  record
    case F is
      when petite=>v:integer range 7..10;
      when others=>NULL;
    end case;
  end record;
type Carte is
  record
    le_niveau:Niveau;
    la_couleur:Couleur;
  end record;
```

Modélisation de données : les cartes à jouer (2/2)

```
dame_pique:Carte:=(dame,pique);
neuf_carreau:Carte:=((petite,9),carreau);
neuf_carreau:Carte:=((petite,9),carreau);
```

Filtrage sur les types articles variants

```
subtype Valeur is Integer range 0..20;
function val_belote (atout:Couleur;c:Carte) return Valeur is
begin
  case c.le_niveau.F is
    when as=>return 11;
    when roi=>return 4;
    when dame=>return 3;
    when valet=>
       if c.la_couleur=atout then return 20; else return
   2;end if;
    when petite=>
      case c.le_niveau.v is
        when 10=> return 10;
        when 9=>
         if c.la_couleur=atout then return 14;else return
   0;end if;
        when others=> return 0;
      end case;
  end case;
end val_belote;
```

Les cartes à jouer : autre solution (1/2)

```
type Couleur is (trefle,carreau,coeur,pique);
type Figure is (as,roi,dame,valet,petite);
type Carte(F:Figure) is
  record
   la_couleur:Couleur;
   case F is
      when petite=>v:Integer range 7..10;
      when others=>NULL;
   end case;
 end record;
```

Les cartes à jouer : autre solution (2/2)

```
un_roi(roi);
--couleur variable
 valet_coeur:Carte:=(valet,coeur);
 une_carte:Carte:=(as,carreau);
 une_carte.v:=7;
-- CONSTRAINT_ERROR
```