Université
Joseph
Fourier

UFR IMAG

Département
Licence Sciences
et Technologie

LICENCE SCIENCES & TECHNOLOGIES
1re année

INF121
ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE

Récursivité et Récurrence

Fonctions récursives

Types récursifs

Ensembles définis récursivement

Principe de récurrence

Table des matières

Chapitre 6

Types récursifs et fonctions récursives

6.1 Ensembles définis récursivement et types récursifs

6.1.1 Exemple introductif : les entiers de Peano

Les éléments de l'ensemble $\mathbb{N}$ sont appelé des entiers naturels car il nous semble naturel (au sens d'habituel) de compter en utilisant les nombres 0,1,2,.... Cependant, **0,1,2,*et cætera*** n'est pas une définition très rigoureuse des entiers.
La définition véritablement mathématique des entiers est due au mathématicien italien Giuseppe Peano (1858-1932). L'ensemble des entiers dits de Peano, noté $\mathbb{P}$ pour faire la distinction ave $\mathbb{N}$, est défini de la manière suivante :

DÉFINITION MATHÉMATIQUE DE L'ENSEMBLE *des entiers de Peano*

déf $\mathbb{P} = \{ \textsc{Zero} \} \cup \{ \textsc{Succ}(p) \mid p \in \mathbb{P} \}$

Remarquez que cette définition est **récursive** puisque la définition de $\mathbb{P}$ fait référence à $\mathbb{P}$:

$$\textbf{\textit{déf}} \quad \boxed{\mathbb{P}} = \{ \textsc{Zero} \} \cup \{ \textsc{Succ}(p) \mid p \in \boxed{\mathbb{P}} \}$$

On dit que l'ensemble $\mathbb{P}$ est défini *récursivement* ou *par réccurence.*

Que dit cette définition ?

1. *Que* Zero *est un entier de Peano*
2. *Que si p est un entier de Peano alors* $\textsc{Succ}(p)$ *(dit le « successeur » de p) est aussi un entier de Peano.*

Une telle définition a-t'elle un sens ? Le mathématicien américain, Stephen Cole Kleene (1909-1994) a donné un sens aux *ensembles définis récursivement* en s'appuyant sur le principe des *suites réccurentes.*

6.1.2 Principe de construction de Kleene

L'ensemble $\mathbb{P}$ est l'union des ensembles $\mathbb{P}_i$ qui sont les termes de la suite récurrente :

$$\begin{cases} \mathbb{P}_0 &= \{\textsc{Zero}\} & \textit{la base de la récurrence} \\ \mathbb{P}_{i+1} &= \{\textsc{Succ}(p) \mid p \in \mathbb{P}_i\} & \textit{le pas de la récurrence} \end{cases}$$

Ainsi,

$$\begin{aligned} \mathbb{P}_1 &= \{\textsc{Succ}(p) \mid p \in \mathbb{P}_0\} = \{\textsc{Succ}(\textsc{Zero})\} \\ \mathbb{P}_2 &= \{\textsc{Succ}(p) \mid p \in \mathbb{P}_1\} = \{\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))\} \\ \mathbb{P}_3 &= \{\textsc{Succ}(p) \mid p \in \mathbb{P}_2\} = \{\textsc{Succ}(\textsc{Succ}(\textsc{Succ}(\textsc{Zero})))\} \\ &\vdots \end{aligned}$$

Et finalement,

$$\begin{aligned}\mathbb{P} &= \mathbb{P}_0 \cup \mathbb{P}_1 \cup \mathbb{P}_2 \cup \dots \\ &= \{\textsc{Zero}\} \cup \{\textsc{Succ}(\textsc{Zero})\} \cup \{\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))\} \cup \dots \\ &= \{\textsc{Zero}, \textsc{Succ}(\textsc{Zero}), \textsc{Succ}(\textsc{Succ}(\textsc{Zero})), \textsc{Succ}(\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))), \dots\}\end{aligned}$$

Théorème *Tout ensemble défini récursivement qui possède un constructeur est forcément infini.* $\mathbb{P}$ *est le plus élémentaire des ensembles définis récursivement puisque il possède une seule entité de base et un seul constructeur récursif et à un seul argument.*

Théorème *Un ensemble défini récursivement est stable par application de ses constructeurs.*

Cela signifie dans le cas de l'ensemble $\mathbb{P}$ que pour tout entier $p \in \mathbb{P}$ si on lui applique le constructeur Succ on obtient un entier $\textsc{Succ}(p)$ qui forcément est déjà dans $\mathbb{P}$. Ainsi, $\mathbb{P}$ ne grossit pas par application des constructeurs : il est stable.

Preuve : Soit un entier $p \in \mathbb{P}$, d'après le principe de construction des termes $\mathbb{P}_i$ de la suite récurrente de S.C.Kleene, l'entier p a été construit à une certaine étape i et donc $p \in \mathbb{P}_i$. Mais alors $\textsc{Succ}(p)$ appartient à $\mathbb{P}_{i+1} \stackrel{def}{=} \{\textsc{Succ}(p) \mid p \in \mathbb{P}_i\}$ et donc $p \in \mathbb{P}$ puisque $\mathbb{P} \stackrel{def}{=} \bigcup \mathbb{P}_i$.

Lien entre $\mathbb{N}$ et $\mathbb{P}$? Intuitivement, Zero correspond à l'entier naturel 0, $\textsc{Succ}(\textsc{Zero})$ correspond au successeur de 0 c'est-à-dire 1, $\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))$ correspond au successeur de 1 c'est-à-dire 2, *etc.* On autorise la notation $\textsc{Succ}^3(\textsc{Zero})$ pour $\textsc{Succ}(\textsc{Succ}(\textsc{Succ}(\textsc{Zero})))$ et de manière générale $\textsc{Succ}^n(\textsc{Zero}) \stackrel{def}{=} \underbrace{\textsc{Succ}(\textsc{Succ}(\dots(\textsc{Succ}}_{n \text{ fois}}(\textsc{Zero})))))$.

6.1.3 Définition du type `peano` correspondant aux entiers de Peano

On peut définir le type CAML correspondant à l'ensemble $\mathbb{P}$ en appliquant le principe vu au chapître 2 de correspondance entre ensemble et un type (voir § **Types somme**).

DÉFINITION MATHÉMATIQUE DE L'ENSEMBLE *des entiers de Peano*

déf $\mathbb{P} = \{\ \textsc{Zero}\ \} \cup \{\ \textsc{Succ}(p) \mid p \in \mathbb{P}\ \}$

DÉFINITION INFORMATIQUE DU TYPE *peano*

```
type peano  = Zero | Succ of peano
```

Remarquez que la définition du type `peano` est récursive exactement là où la définition de l'ensemble $\mathbb{P}$ était récursive : `type` `peano` `= Zero | Succ of` `peano`

Comment lire cette définition de type ?

Une donnée de type `peano` est soit de la forme Zero, soit de la forme $\textsc{Succ}(p)$ où p est de type `peano`.

Les éléments du type `peano` sont donc exactement les entiers de l'ensemble $\mathbb{P}$.

- Zero est une **constante symbolique**, c'est la base du type `peano`, elle a le type Zero : `peano`
- Succ est un **constructeur**, il a le type Succ : `peano` $\rightarrow$ `peano` qui indique que Succ permet de construire un nouvel entier $S(p)$ de type `peano` à partir d'un entier p de type `peano`.

Remarques

- Il n'est pas possible d'utiliser les opérateurs arithmétiques `+`, `-`, `*`, `/`, `quo`, `mod`, ... avec des entiers de type `peano` car ces opérateurs exigent des arguments de type `int`. En TD et TP vous verrez comment redéfinir les opérateurs arithmétiques de base pour les entiers de Peano.
- Le type `peano` est prédéfini dans le module `Inf121`.
- ZERO n'est pas l'entier 0, c'est une constante symbolique. En revanche, il est aisé de convertir un entier `peano` en `int` (voir exercice ci-après).
- Quelques exemples extraits d'une session CAML lancée par `ledit ocaml`

```
                Objective Caml version 3.11.1
# ZERO ;;
Error: Unbound constructor ZERO

# type peano  = ZERO | SUCC of peano  ;;
type peano  = ZERO | SUCC of peano  is defined

# ZERO ;;
- : peano  = ZERO
# 0 ;;
- : int = 0
# ZERO = 0 ;;
      ^
 Error: This expression has type int but an expression
        was expected of type peano

# SUCC(ZERO) ;;
- : peano  = SUCC ZERO

# SUCC(ZERO) = ZERO + 1 ;;
         ^
Error: This expression has type peano  but an expression
       was expected of type int
```

6.2 Fonctions opérant sur un type récursif

6.2.1 Conversion entre entiers de Peano et entiers naturels

Imaginons qu'on veuille écrire une fonction *p_vers_n* qui convertit un entier de Peano en entier naturel :

SPÉCIFICATION MATHÉMATIQUE

Profil $p\_vers\_n \ : \ \mathbb{P} \to \mathbb{N}$
Sémantique : $p\_vers\_n(p)$ *est l'entier naturel correspondant à l'entier de Peano* p

Exemples

1. $p\_vers\_n(\text{ZERO}) = \ldots$
2. $p\_vers\_n(\text{SUCC}(\text{SUCC}(\text{ZERO}))) = 2$

Propriété $\forall n \in \mathbb{N},\ p\_vers\_n(\text{SUCC}^n(\text{ZERO})) = n$

Comment définir la fonction *pvn* **?** D'après la définition du type `peano`, un entier `peano` est soit de la forme ZERO, soit de la forme SUCC(p). Puisque ce sont les deux seules manières de constuire une donnée de type `peano`, pour définir la fonction *p_vers_n* il suffit d'indiquer ce qu'elle fait dans le cas

ZERO et ce qu'elle fait dans le cas SUCC(p). La fonction sera donc complétement définie si on complète les équations suivantes :

Définition récursive de la fonction p_vers_n par des équations

1) $p\_vers\_n(\text{ZERO}) = ..$ *équation du cas de base*

2) $p\_vers\_n(\text{SUCC}(p)) = ... p\_vers\_n(p)$ *équation du constructeur récursif*

Remarquez que la fonction p_vers_n est récursive exactement là où le type `peano` était récursif :

`type` `peano` `=` `Zero` `|` `Succ of` `peano`

La récursivité apparaît dans le cas du constructeur SUCC ; l'équation (2) qui traite le cas SUCC sera donc forcément récursive, c'est-à-dire que le calcul de $p\_vers\_n(\text{SUCC}(p))$ se fait en utilisant le résulat du calcul de $p\_vers\_n(p)$.

$$\textit{équation 2)}\quad \boxed{p\_vers\_n(\overbrace{\text{SUCC}(p)}^{\texttt{peano}})} = ... \boxed{\underbrace{p\_vers\_n(\overbrace{p}^{\texttt{peano}})}_{\mathbb{N}}}$$

Remarque L'équation (2) utilise en quelque sorte une raisonnement par récurrence qui découle de la définition récursive du type peano. Pour définir le résulat de $p\_vers\_n(\text{SUCC}(p))$ on suppose qu'on dispose du résulat de $p\_vers\_n(p)$. Notez qu'on a le droit d'appliquer p_vers_n à p puisque par définition de SUCC(p) on sait que p est un entier de peano et on sait, d'après la spécification de p_vers_n que $p\_vers\_n(p)$ sera alors l'entier naturel correspondant à l'entier de Peano p. Pour compléter les pointillés de l'équation (2) il suffit alors d'expliquer comment on passe de l'entier naturel $p\_vers\_n(p)$ à l'entier naturel correspondant à $p\_vers\_n(\text{SUCC}(p))$: il suffit de faire +1.

Implantation en caml Une fois que les équations sont complétées, il est aisé de traduire les équations sous forme d'une fonction CAML qui sera forcément récursive exactement là où les équations sont récursives.

```
let rec (p_vers_n : peano  -> ......) =
  function
    | ......... -> 0                          (* équation 1 *)

    | Succ(p) -> 1 + .....................   (* équation 2 *)
;;
```

Exercice : conversion des entiers naturels en entiers de Peano

On chercher à définir (c'est-à-dire donner la spécification, équations, implantation) la fonction n_vers_p qui transforme un entier naturel en entier de Peano.

SPÉCIFICATION MATHÉMATIQUE

Profil n_vers_p : .... $\rightarrow$ ...

Sémantique : $n\_vers\_p(n)$ *est l'entier de Peano correspondant à l'entier naturel* n

Exemples

1. $n\_vers\_p(0) =$ .....
2. $n\_vers\_p(1) = \text{SUCC}(\text{ZERO})$

Propriété $\forall n \in \mathbb{N},\ n\_vers\_p\,(n) = \text{SUCC}^n(\text{ZERO}))$

RÉALISATION INFORMATIQUE

Définition récursive de la fonction par des équations

1) $n\_vers\_p\,(0) \quad = \quad \ldots\ldots\ldots\ldots$

2) $n\_vers\_p\,(n) \quad = \quad \ldots\ldots\ldots\ldots(\ n\_vers\_p\,(\ldots\ldots\ldots\ldots)\ )$ ***quand*** $n \ldots 0$

Implantation

```
let rec (n_vers_p : ..... -> peano) =
  function
    | 0 -> .........

    | n when n..0 -> Succ( ................( n-1 ))
```

CAML signale que le filtrage n'est pas exhaustif : en effet le cas `n<0` n'est pas considéré. Ce n'est pas une erreur, au contraire c'est volontaire : cela indique que la fonction *n_vers_p* n'est pas définie pour $n < 0$ puisqu'il n'existe pas d'entier de Peano correspondant à un entier négatif.

6.2.2 Testeurs et sélecteurs pour le type `peano`

Un entier de Peano est soit de la forme ZERO, soit de la forme SUCC(p) c'est-à-dire le successeur d'un entier de Peano p. Pour distinguer ces deux cas au niveau de la réalisation d'une fonction, on implante une analyse par cas :

SPÉCIFICATION MATHÉMATIQUE *testeur sur* $\mathbb{P}$

Profil $est\_nul\ :\ \mathbb{P} \rightarrow Bool$

Sémantique : *est_nul (p) teste si l'entier de Peano p est l'entité de base* ZERO.

RÉALISATION INFORMATIQUE

```
let (est_nul : peano  -> bool) = function p -> p=Zero ;;
```

Lorsqu'un entier de Peano n'est pas ZERO, il s'écrit nécessairement SUCC(p) où SUCC(p) est appelé le « successeur » de p et inversement p est appelé le « prédécesseur » de SUCC(p).

SPÉCIFICATION MATHÉMATIQUE *sélecteur dans* $\mathbb{P}$

Profil $prédécesseur\ :\ \mathbb{P} \setminus \{\ldots\ldots\ldots\ldots\} \rightarrow \mathbb{P}$

Sémantique : *prédécesseur (p) est l'entier de Peano qui précède p.*

Exemple $prédécesseur\,(\text{SUCC}(\text{SUCC}(\text{ZERO}))) = \text{SUCC}(\text{ZERO})$

Propriété $\forall p \in \mathbb{P}, prédécesseur\,(\text{SUCC}(p)) = p$

RÉALISATION INFORMATIQUE

Implantation

```
let (prédécesseur : peano  -> ..........) =
   function
     | Succ(p) -> p
```

Remarquez qu'on a volontairement oublié le cas ZERO du filtrage puisque la fonction *prédécesseur* n'est pas définie pour ZERO. CAML signale que le filtrage n'est pas exhaustif (on a oublié le cas ZERO)

> Warning P : this pattern-matching is not exhaustive. Here is an example of a value that is not matched : ZERO

Ce message de CAML est utile pour rappeler aux utilisateurs que la fonction *prédécesseur* n'est pas définie pour ZERO.

6.3 Principe de récurrence associé à un type récursif

On va voir qu'à chaque définition de type récursif correspond un principe de récurrence. Vous connaissez déjà le principe de récurrence sur les entiers :

Pour montrer qu'une propriété *Prop* est vraie pour tout entiers $n \in \mathbb{N}$

$$\frac{\overbrace{Prop(0)}^{on\ montre} \quad \overbrace{Prop(n) \Rightarrow Prop(n+1)}^{on\ montre}}{\forall n \in \mathbb{N},\ Prop(n)} \leftarrow \textit{cette ligne horizontale se lit « on en déduit » par récurrence sur } \mathbb{N}$$

nous allons voir que le principe de preuve par récurrence découle directement du principe de construction de Kleene appliqué aux entiers de Peano.

Commençons par rappeler qu'une propriété mathématique (on dit aussi *prédicat*) est une fonction qui retourne un booléen. Une propriété *Prop* sur les entiers de Peano est donc une fonction qui a pour profil $Prop : \mathbb{P} \rightarrow Bool$.

Comment démontrer qu'une propriété *Prop* est satisfaite par tout entier de Peano ?

D'après la définition ***déf*** $\mathbb{P} = \{ \text{ZERO} \} \cup \{ \text{SUCC}(p) \mid p \in \mathbb{P} \}$ on constate qu'il y a seulement deux sortes d'entiers de Peano : ZERO et ceux qui sont de la forme $\text{SUCC}(p)$ où $p \in \mathbb{P}$.

Pour montrer que la propriété *Prop* est vraie pour tout entier de Peano, c'est-à-dire $\forall p \in \mathbb{P}, Prop(p)$ il suffit donc de montrer :

- **(démontration D1)** que la propriété *Prop* est vraie pour ZERO, c'est-à-dire montrer :

  (D1) $Prop(\text{ZERO})$
- **(démontration D2)** que si la propriété est vraie pour p alors elle est vraie pour l'entier suivant, $\text{SUCC}(p)$ c'est-à-dire montrer :

  (D2) $Prop(\boxed{p}) \Rightarrow Prop(\text{SUCC}(\boxed{p}))$

Remarquez qu'on retrouve pour les entiers de Peano le principe de récurrence que vous connaissez sur $\mathbb{N}$. Mais il y a plus intéressant, on est maintenant capable d'expliquer d'où vient ce principe de preuve par récurrence : c'est une conséquence de la construction de Kleene (voir § 6.1.2).

$$\begin{array}{rcccccccc}
\mathbb{P} & = & \mathbb{P}_0 & \cup & \mathbb{P}_1 & \cup & \mathbb{P}_2 & \cup & \ldots \\
 & = & \{\text{ZERO}\} & \cup & \{\text{SUCC}(\text{ZERO})\} & \cup & \{\text{SUCC}(\text{SUCC}(\text{ZERO}))\} & \cup & \ldots
\end{array}$$

$$\overbrace{Prop(\text{ZERO})}^{\text{d'après D1}}$$

$$\text{or} \quad \underbrace{Prop(\boxed{\text{ZERO}}) \Rightarrow Prop(\text{SUCC}(\boxed{\text{ZERO}}))}_{\text{d'après D2 appliqué dans le cas où } p \text{ est ZERO}}$$

$$\text{d'où} \quad Prop(\text{SUCC}(\text{ZERO}))$$

$$\text{or} \quad \underbrace{Prop(\boxed{\text{SUCC}(\text{ZERO})}) \Rightarrow Prop(\text{SUCC}(\boxed{\text{SUCC}(\text{ZERO})}))}_{\text{d'après D2 appliqué dans le cas où } p \text{ est SUCC(ZERO)}}$$

$$\text{d'où} \quad Prop(\text{SUCC}(\text{SUCC}(\text{ZERO})))$$

et ainsi de suite

Finalement, si on a réussi à démontrer $D1$ et $D2$ alors en utilisant une fois le résultat de la démontration $D1$ et autant de fois que nécessaire le résultat de la démonstration $D2$ on peut montrer que la propriété *Prop* est vrai pour n'importe quel entier de Peano.

$$\begin{array}{rcccccccc}
\mathbb{P} & = & \mathbb{P}_0 & \cup & \mathbb{P}_1 & \cup & \mathbb{P}_2 & \cup & \ldots \\
& = & \{\textsc{Zero}\} & \cup & \{\textsc{Succ}(\textsc{Zero})\} & \cup & \{\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))\} & \cup & \ldots \\
& & \underbrace{\mathit{Prop}(\textsc{Zero})}_{D1} & \underbrace{\Rightarrow}_{D2} & \mathit{Prop}(\textsc{Succ}(\textsc{Zero})) & \underbrace{\Rightarrow}_{D2} & \mathit{Prop}(\textsc{Succ}(\textsc{Succ}(\textsc{Zero}))) & \underbrace{\Rightarrow}_{D2} & \mathit{Prop}(\ldots)
\end{array}$$

Le principe de récurrence pour les entiers de Peano est une conséquence directe du principe de construction de Kleene.

$$\frac{\overbrace{\mathit{Prop}(\textsc{Zero})}^{\textit{on montre}} \qquad \overbrace{\mathit{Prop}(p) \Rightarrow \mathit{Prop}(\textsc{Succ}(p))}^{\textit{on montre}}}{\forall p \in \mathbb{P},\ \mathit{Prop}(p)}\ \textit{on en déduit par récurrence sur } \mathbb{P}$$

Le principe de récurrence sur $\mathbb{N}$ cache en fait le principe de récurrence sur $\mathbb{P}$ où on note 0 à la place de Zero et $p+1$ au lieu de $\textsc{Succ}(p)$.

6.4 Généralisation

6.4.1 Types récursifs avec des constructeurs d'arité quelconque

Considérons le cas d'un ensemble *EnsRec* défini récursivement avec n entité de bases $\mathrm{B}_1, \ldots, B_n$, un constructeur unaire Cu (à un argument) un constructeur binaire Cb (à deux arguments).

DÉFINITION MATHÉMATIQUE D'UN ENSEMBLE

$$\begin{array}{rrcl}
\textbf{\textit{déf}} & \mathit{EnsRec} & = & \{\ \mathrm{B}_1, \ldots, \mathrm{B}_n\ \} \\
& & \cup & \{\ \textsc{Cu}(e) \mid e \in \mathit{EnsRec}\ \} \\
& & \cup & \{\ \textsc{Cb}(e_1, e_2) \mid e_1, e_2 \in \mathit{EnsRec}\ \}
\end{array}$$

Les concepts présentés dans le cas des entiers de Peano se généralisent. La définition de l'ensemble E est récursive en $\textsc{Cu}(\boxed{e}), \textsc{Cb}(\boxed{e_1}, \boxed{e_2})$ puisque $e, e_1, e_2 \in E$. On peut traduire cette définition d'ensemble en un type CAML qui sera récursif pour les arguments de Cu et Cb.

DÉFINITION INFORMATIQUE DU TYPE

```
type ens_rec = B1 | ... | Bn
     | Cu of ens_rec
     | Cb of ens_rec * ens_rec
```

Toute fonction récursive $f : \mathit{EnsRec} \rightarrow \ldots$ qui opére sur des données de type `ens_rec` sera définie par une équations pour chacun des constantes et des constructeurs et sera récursive exactement là où le type est récursif. Les équations seront donc de la forme :

Définition récursive de la fonction f par des équations

$$\begin{array}{rcl}
f(\mathrm{B}_1) & = & \ldots \\
& \vdots & \\
f(\mathrm{B}_n) & = & \ldots \\
f(\textsc{Cu}(e)) & = & \ldots\ f(e)\ \ldots \\
f(\textsc{Cb}(e_1, e_2)) & = & \ldots\ f(e_1)\ \ldots\ f(e_2)\ \ldots
\end{array}$$

Le principe de récurrence associé au type Pour montrer qu'une propriété $Prop : EnsRec \rightarrow Bool$ est vraie pour tout élément de type `ens_rec`, on dispose du principe de récurrence qui découle automatiquement de la définition du type `ens_rec` :

$$\frac{\overbrace{Prop(\mathrm{B}_1)}^{on\ montre} \quad \ldots \quad \overbrace{Prop(\mathrm{B}_n)}^{on\ montre} \quad \overbrace{Prop(e) \Rightarrow Prop(\mathrm{Cu}(e))}^{on\ montre} \quad \overbrace{Prop(e_1) \ \wedge \ Prop(e_2) \Rightarrow Prop(\mathrm{Cb}(e_1, e_2))}^{on\ montre}}{\forall e \in EnsRec,\ Prop(e)}$$

6.4.2 Exemple complexe : les expressions arithmétiques

DÉFINITION MATHÉMATIQUE D'UN ENSEMBLE

$$\begin{aligned} \textbf{\textit{déf}} \quad ExprArith \ &= \ \{\ \mathrm{I}(n), +\infty, -\infty, \mathrm{Var}(s) \mid n \in \mathbb{Z},\ s \in Chaîne\ \} \\ &\cup \ \{\ \mathrm{Neg}(e) \mid e \in ExprArith\ \} \\ &\cup \ \{\ \mathrm{Add}(e_1, e_2), \mathrm{Mul}(e_1, e_2) \mid e_1, e_2 \in ExprArith\ \} \end{aligned}$$

DÉFINITION INFORMATIQUE DU TYPE

```
type expr_arith =
  | I of int | Pinf | Minf | Var of string  ..................... les entités de base
  | Neg of expr_arith  ............................. un constructeur récursif unaire
  | Add of expr_arith * expr_arith  ........ un premier constructeur récursif binaire
  | Mul of expr_arith * expr_arith  ......... un second constructeur récursif binaire
```

Le type `expr_arith` permet de représenter l'expression arithmétique $x + (y - 1) * x - y * x$ en CAML par la donnée de type `expr_arith`

Add(Var("x"), Add(Mul(Add(Var("y") , I(-1)) , Var("x")), Neg(Mul(Var("y"), Var("x")))))

Il est alors possible de manipuler les expressions arithmétiques par exemple pour écrire une fonction de simplification automatique qui prend en paramètre une `expr_arith` et retourne une `expr_arith` simplifiée. Dans l'exemple précédent, la fonction devrait retourner I(0).

La simplification d'expression arithmétique fait régulierement partie du sujet d'examen.

6.4.3 Exemple de fonction récursive sur les expressions arithmétiques

SPÉCIFICATION MATHÉMATIQUE

Profil *affiche* : $ExprArith \rightarrow Chaîne$

Sémantique : *affiche*(ea) *est la chaîne de caractères correspondant à la notation mathématique de l'expression arithmétique.*

RÉALISATION INFORMATIQUE

```
let (par: string -> string) = function str -> "(" ^ str ^ ")" ;;

let (est_addition: expr_arith -> bool) =
  function
    | Add(e1,e2) -> .........

    | _ -> ............
;;

let rec (affiche: expr_arith -> string) =
  function
```

```
    | I(n) -> string_of_int n
    | PINF -> "+∞"
    | MINF -> "-∞"
    | VAR(s) -> s
    | NEG(e) -> "-" ^ par(affiche(e))
    | ADD(e1,e2) -> affiche(e1) ^ " + " ^ affiche(e2)
    | MUL(e1,e2) ->
                (if est_addition(e1) then par(affiche(e1)) else affiche(e1))
                ^ " * " ^
                (if ..................................... then .....................................

                 else ........................ )
;;
```

Pour l'expression donnée en exemple on obtient l'affichage `x + (y + -1) * x + -(y * x)`. En guise d'exercice on laisse le soin au lecteur d'améliorer cet affichage en s'inspirant de ce que l'on a fait pour le constructeur MUL.

6.4.4 Propriétés démontrables par récurrence

Propriété *Toute expression de type* `expr_arith` *avec v variables et c constantes contient exactement $v + c - 1$ constructeurs* ADD *ou* MUL. *Ce qui s'exprime mathématiquement par la propriété*

SPÉCIFICATION MATHÉMATIQUE

Profil $Prop : ExprArith \to Bool$

$$Prop(e) \stackrel{def}{=} \left(\underbrace{|e|_{\text{ADD}}}_{a} + \underbrace{|e|_{\dots\dots\dots}}_{m} = \underbrace{|e|_{\dots\dots\dots}}_{v} + \underbrace{|e|_{\text{PINF}} + |e|_{\dots\dots\dots\dots\dots} + |e|_{\text{I}}}_{c} - 1 \right)$$

où $|e|_{constructeur}$ est le nombre d'apparitions du *constructeur* dans l'expression e

Exercice Prouvez la propriété *Prop* pour toute expression de type *ExprArith* en utilisant le principe de récurrence associé à *ExprArith*.

$$\frac{\overbrace{Prop(\text{I}(n))}^{on\ montre} \quad \overbrace{Prop(\dots\dots\dots)}^{on\ montre} \quad \overbrace{Prop(\text{MINF})}^{on\ montre} \quad \overbrace{Prop(\dots\dots\dots\dots\dots)}^{on\ montre}}{\dfrac{\overbrace{Prop(e) \Rightarrow Prop(\dots\dots\dots\dots\dots)}^{on\ montre}}{\dfrac{\overbrace{Prop(e_1) \ \wedge \ Prop(e_2) \Rightarrow Prop(\text{ADD}(e_1, e_2))}^{on\ montre}}{\dfrac{\overbrace{Prop(e_1) \ \wedge \ Prop(e_2) \Rightarrow Prop(\dots\dots\dots(e_1, e_2))}^{on\ montre}}{\forall e \in ExprArith, \ Prop(e)}\ on\ en\ déduit\ par\ récurrence\ sur\ ExprArith}}}$$