Chapitre 3

Définitions récursives et induction structurelle

Plan

1. Définitions récursives

2. Induction structurelle

3. Exemples
   - Arbres
   - Naturels
   - Expressions arithmétiques

Lectures conseillées :

- MCS : chapitre 6

Définition récursive

Une *définition récursive* explique comment construire un élément d'un type donné à partir d'élément du même type plus simple.

Un *type de données récursif* R est défini par :

- des règles de base qui affirment que des éléments appartiennent à R
- des règles inductives de construction de nouveaux éléments de R à partir de ceux déjà construits
- Une règle (souvent implicite) qui dit que seuls les éléments obtenus par les deux règles précédentes sont dans R.

Exemples : Soit l'ensemble E des entiers pairs. E peut être défini comme suit :

- Cas de base : $0 \in E$,
- Cas récursifs :
  1. si $n \in E$, alors $n + 2 \in E$,
  2. si $n \in E$, alors $-n \in E$.

Chaîne de caractères

Définition : Soit A un ensemble non vide appelé *alphabet*, dont les éléments sont appelés des caractères, lettres ou symboles. L'ensemble A^* des chaînes (*string*) sur l'alphabet A est défini récursivement comme suit :

- Cas de base : La chaîne vide, λ, est dans A^*.
- Cas récursif : Si a $\in A$ et $x \in A^*$, alors a$x \in A^*$.
  (où ax désigne la chaîne x au début de laquelle on a ajouté le caractère a).

Exemple : Si $A = \{0, 1\}$, λ, 1, 1011 et 00101 appartiennent à A^*.

Définition sur un ensemble défini récursivement

Des fonctions ou opérations peuvent être définies, récursivement, sur les éléments d'un ensemble défini de manière récursive.

Définition : La *longeur* $|x|$ d'une chaîne est *définie récursivement* sur base de la définition de $x \in A^*$:

- Cas de base : $|\lambda| ::= 0$.[4]
- Cas récursif : Si $x \in A^*$ et $a \in A$, $|ax| ::= 1 + |x|$.

Exemple : $|1011| = 1 + |011| = 2 + |11| = 3 + |1| = 4 + |\lambda| = 4$

Définition : Soit deux chaînes x et $y \in A^*$, la concaténation $x \cdot y$ de x et y est définie récursivement sur base de la définition de $x \in A^*$:

- Cas de base : $\lambda \cdot y ::= y$
- Cas récursif : Si $x \in A^*$ et $a \in A$, $ax \cdot y ::= a(x \cdot y)$.

On notera dans la suite $x \cdot y$ par xy.

Exemple : $01 \cdot 010 = 0(1 \cdot 010) = 0(1(\lambda \cdot 010)) = 0(1(010)) = 01010$.

[4] ::= signifie "égal par définition".

Preuve sur un ensemble défini récursivement

Définition : Soit $A = \{0, 1\}$ et soit $S \subset A^*$ l'ensemble défini récursivement comme suit :

- Cas de base : $\lambda \in S$, où λ est la chaîne vide
- Cas récursifs :
  1. Si $x \in S$, alors $0x1 \in S$,
  2. Si $x \in S$, alors $1x0 \in S$,
  3. Si $x, y \in S$, alors $xy \in S$, où xy désigne la concaténation de x et y.

Théorème : Tout élément s dans S contient le même nombre de 0 et de 1.

(Exercice : Montrez que toute chaîne $s \in A^*$ contenant le même nombre de 0 et de 1 appartient à S. En déduire que S est l'ensemble de toutes les chaînes de A^* contenant autant de 0 que de 1.)

Démonstration : Soit le prédicat $P(n)$ vrai si toute chaîne s de S obtenue après n applications des régles récursives contient un nombre identique de 0 et de 1. Montrons que $P(n)$ est vrai pour tout $n \in \mathbb{N}$ par *induction forte*.
Cas de base ($n = 0$) : Le seul élément qui peut être obtenu après 0 applications des règles récursives est la chaîne vide qui contient un nombre identique de 0 et 1.
Cas inductif ($n \geq 0$) : Supposons $P(0), \ldots, P(n)$ vérifiés et montrons que $P(n+1)$ l'est également. Soit s une séquence obtenue après $n+1$ applications des règles. Il y a 3 cas possibles :

- Cas 1 : la $n+1$ règle est la règle 1. On a $s = 0x1$, où x est obtenu après n applications des règles et contient donc le même nombre de 0 et de 1 (par hypothèse inductive). s contient donc également le même nombre de 0 et de 1 (un de plus que x).
- Cas 2 : la $n+1$ régle est la règle 2. Ce cas est symétrique au cas précédent.
- Cas 3 : la $n+1$ règle est la règle 3. $s = xy$ où x et y sont dans S et ont été obtenus par au plus n applications des règles. x et y contenant un nombre identique de 0 et de 1, c'est aussi le cas de $s = xy$. □

Induction structurelle

Le schéma de la démonstration précédente est applicable à tous les types de données définis récursivement. On peut se passer de l'induction sur le nombre d'applications des règles en invoquant le *principe d'induction structurelle*.

Principe d'induction structurelle : Soit un prédicat P défini sur un type de données R défini récursivement. Pour montrer que P est vrai pour tout élément de R, il suffit de montrer

- que P est vrai pour chaque élément de base, $b \in R$ et
- que P est vrai pour tout élément résultant de l'application d'une règle récursive, en supposant qu'il est vrai pour les éléments combinés par cette règle.

Le principe d'induction structurelle peut se montrer par induction forte sur le nombre d'applications des règles récursives.

Exemple

Pour montrer qu'un prédicat P est vrai pour tout élément de S (transp. 119), il faut montrer :

- que $P(\lambda)$ est vrai et
- que $P(0x1)$, $P(1x0)$ et $P(xy)$ sont vrais dès que $P(x)$ et $P(y)$ sont vrais.

Réécriture de la démonstration du transp. 120 :
Soit $P(s) =$"s contient autant de 0 que de 1". Montrons pas induction structurelle que $P(s)$ est vrai pour tout $s \in S$.

Cas de base : $P(\lambda)$ est vrai trivialement.
Cas inductifs : Si x et y contiennent autant de 1 que de 0, c'est aussi le cas de $1x0$, $0x1$ et xy.

Par le principe d'induction structurelle, on en conclut que $P(s)$ est vrai pour tout $s \in S$. □

Un exemple (un peu) plus compliqué

Définition : Soit un sous-ensemble de chaînes de caractères sur l'alphabet $\{\mathrm{M}, \mathrm{I}, \mathrm{U}\}$ défini de manière récursive comme suit, et appelé le système MIU :

- Cas de base : MI appartient au système MIU
- Cas récursifs :
  1. Si xI est dans le système MIU, où x est une chaîne, alors xIU est dans le système MIU.
  2. Si Mx est dans le système MIU, où x est une chaîne, alors Mxx est le système MIU.
  3. Si xIIIy est dans le système MIU, où x et y sont deux chaînes (potentiellement vides), alors xUy est dans le système MIU.
  4. Si xUUy est dans le système MIU, où x et y sont deux chaînes (potentiellement vides), alors xUy est dans le système MIU.

MUIU appartient-il au système MIU ? Qu'en est-il de MU ?

Source : Douglas Hofstadter, *Gödel, Escher, Bach* (New York : Basics Books), 1979.

Appartenance

Théorème : MUIU appartient au système MIU.

Démonstration :
Montrons qu'on peut dériver MUIU du cas de base MI en utilisant les règles récursives :

- Par le cas de base, MI appartient au système MIU.
- En utilisant la règle 2, on en déduit que MII $\in$ au système MIU.
- En utilisant la règle 2, on en déduit que MIIII $\in$ au système MIU.
- En utilisant la règle 3, on en déduit que MUI $\in$ au système MIU.
- En utilisant la règle 1, on en déduit que MUIU $\in$ au système MIU.

□

Prédicat (invariant)

Théorème : Le nombre de I dans une chaîne s appartenant au système MIU n'est jamais un multiple de 3.
Démonstration : Soit $P(s)$ ="le nombre de I dans s n'est pas un multiple de 3". Montrons par induction structurelle que $P(s)$ est vrai pour toute chaîne du système MIU.
Cas de base : Le nombre de I dans MI n'est pas une multiple de 3.
Cas récursifs :

- Règle 1 : Par hypothèse inductive, on suppose que le nombre de I dans xI n'est pas un multiple de 3. Le nombre de I dans xIU n'est donc pas non plus un multiple de 3.
- Règle 2 : Par hypothèse inductive, on peut supposer que le nombre de I dans Mx est soit $3k+1$, soit $3k+2$ pour un k. Le nombre de I dans Mxx est donc soit $6k+2 = 3(2k)+2$, soit $6k+4 = 3(2k+1)+1$. Aucun des deux n'est donc un multiple de 3.

- Règle 3 : Par hypothèse inductive, on peut supposer que le nombre de I dans xIIIy est soit $3k+1$, soit $3k+2$ pour un k. Le nombre de I dans xUy est soit $3(k-1)+1$, soit $3(k-1)+2$, aucun n'étant un multiple de 3.
- Règle 4 : Par hypothèse inductive, on suppose que le nombre de I dans xUUy n'est pas un multiple de 3. Le nombre de I dans xUy n'est donc pas non plus un multiple de 3.

Par induction structurelle, on peut en conclure que $P(s)$ est vérifié pour tout s du système MIU. □

Corrolaire : MU n'appartient pas au système MIU.
Démonstration : C'est une conséquence directe du théorème précédent puisque MU contient zéro I qui est un multiple de 3. □

Modélisation par une machine d'état

Soit la machine d'état $\mathcal{M} = (\mathcal{Q}, \mathcal{Q}_0, \delta)$ définie comme suit :

- $\mathcal{Q} = A^*$ où $A = \{\mathsf{M}, \mathsf{I}, \mathsf{U}\}$,
- $\mathcal{Q} = \{\mathsf{MI}\}$,
-

$$\begin{aligned}
\delta = \quad & \{x\mathsf{I} \rightarrow x\mathsf{IU} | x \in A^*\} \\
\bigcup \quad & \{\mathsf{M}x \rightarrow \mathsf{M}xx | x \in A^*\} \\
\bigcup \quad & \{x\mathsf{III}y \rightarrow x\mathsf{U}y | x, y \in A^*\} \\
\bigcup \quad & \{x\mathsf{UU}y \rightarrow x\mathsf{U}y | x, y \in A^*\}
\end{aligned}$$

L'ensemble des états atteignables par $\mathcal{M}$ correspond exactement au système MIU.

Le prédicat $P(s) =$"le nombre de I dans s n'est pas un multiple de 3" est un invariant de la machine d'état. $Q(s) =$"$s \neq$ MU" découle directement de $P(s)$.

Ambiguité d'une définition récursive

Pour prouver qu'un élément r appartient à un ensemble défini de manière récursive R, il suffit de trouver un ordre d'applications des règles à partir d'un ou plusieurs éléments de base qui permet de générer r.

Exemple : $0110 \in S$ car $\lambda \longrightarrow 01$ (règle 1), $\lambda \longrightarrow 10$ (règle 2), et $01, 10 \longrightarrow 0110$ (règle 3).

S'il existe plusieurs dérivations d'un même élément, on dira que la définition est *ambiguë*.

Exemple : La définition de S est ambiguë. Par exemple, 1010 peut être dérivé de deux manières :

- $\lambda \longrightarrow 10$ (règle 2), puis $10, 10 \longrightarrow 1010$ (règle 3)
- $\lambda \longrightarrow 01$ (règle 1), puis $01 \longrightarrow 1010$ (règle 2)

Ambiguité d'une définition récursive

Une fonction définie sur base d'une définition ambiguë peut être mal définie.

Définition : Soit la fonction $f : S \rightarrow \mathbb{N}$ calculant le nombre d'applications de la règle 3 pour générer s :

- Cas de base : $f(\lambda) = 0$
- Cas récursifs :
  1. $f(0x1) = f(x)$
  2. $f(1x0) = f(x)$
  3. $f(xy) = 1 + f(x) + f(y)$

La définition de f mène à une contradiction :

$$\begin{aligned} f(1010) &= 1 + f(10) + f(10) = 1 + f(\lambda) + f(\lambda) = 1 \\ &= f(01) = f(\lambda) = 0 \end{aligned}$$

(Exercice : trouver une définition récursive de S non ambiguë)

Plan

1. Définitions récursives

2. Induction structurelle

3. Exemples
   - Arbres
   - Naturels
   - Expressions arithmétiques

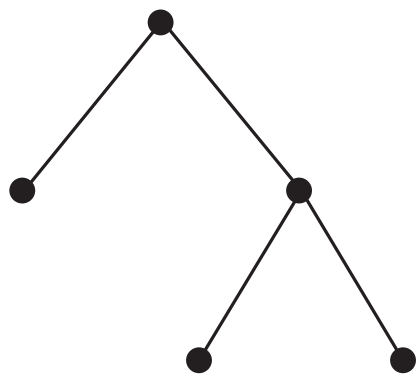
Arbres

On peut définir récursivement une structure d'arbre binaire entier.

Définition : L'ensemble *bTree* des arbres binaires entiers est défini comme suit :

- Cas de base : La feuille **leaf** appartient à *bTree*.
- Cas récursif : Soit $T_1, T_2 \in bTree$, alors **branch**$(T1, T2) \in bTree$.

Exemple : **branch**(**leaf**, **branch**(**leaf**, **leaf**)) représente l'arbre ci-dessous.



Définition récursive sur un arbre

Définitions : La fonction leaves : $bTree \to \mathbb{N}$ compte le nombre de feuilles dans un arbre :

- Cas de base : leaves(**leaf**) ::= 1
- Cas récursif : leaves(**branch**(T_1, T_2)) ::= leaves(T_1) + leaves(T_2)

La fonction branches : $bTree \to \mathbb{N}$ compte le nombre de branches dans un arbre :

- Cas de base : branches(**leaf**) ::= 0
- Cas récursif :
  branches(**branch**(T_1, T_2)) ::= branches(T_1) + branches(T_2) + 1

Théorème : Pour tout arbre $T \in bTree$,

$$\text{leaves}(T) = \text{branches}(T) + 1.$$

Démonstration : Par induction structurelle sur T :

- Cas de base : Par définition, $\text{leaves}(\textbf{leaf}) = 1 = 1 + \text{branches}(\textbf{leaf})$.
- Cas récursif : Par définiton,

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{leaves}(T_1) + \text{leaves}(T_2).$$

Par hypothèse inductive, on a :

$$\text{leaves}(T_1) = \text{branches}(T_1) + 1, \ \text{leaves}(T_2) = \text{branches}(T_2) + 1$$

et donc

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{branches}(T_1) + 1 + \text{branches}(T_2) + 1.$$

Par définition,

$$\text{branches}(\textbf{branch}(T_1, T_2)) = \text{branches}(T_1) + \text{branches}(T_2) + 1$$

et donc :

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{branches}(\textbf{branch}(T_1, T_2)) + 1.$$

□

Exercice

Définitions : La fonction nodes : $bTree \to \mathbb{N}$ calcule le nombre total de nœuds de l'arbre :

- Cas de base : nodes(**leaf**) ::= 1
- Cas récursif : nodes(**branch**(T_1, T_2)) ::= nodes(T_1) + nodes(T_2) + 1

La fonction height : $bTree \to \mathbb{N}$ calcule la hauteur d'un arbre comme suit :

- Cas de base : height(**leaf**) ::= 0
- Cas récursif :
  height(**branch**(T_1, T_2)) ::= 1 + max(height(T_1), height(T_2))

Démontrez le théorème suivant :

Théorème : Pour tout arbre $T \in bTree$,

$$\text{nodes}(T) \leq 2^{\text{height}(T)+1} - 1.$$

Fonctions récursives sur les entiers non-négatifs

Définition : L'ensemble $\mathbb{N}$ est un type de données défini récursivement comme suit :

- Cas de base : $0 \in \mathbb{N}$.
- Cas récursif : si $n \in \mathbb{N}$, alors le successeur, $n+1$, de n est dans $\mathbb{N}$.

L'induction ordinaire est donc équivalente à l'induction structurelle sur la définition récursive de $\mathbb{N}$.

Cette définition récursive permet aussi de définir des fonctions récursives sur les naturels.

Exemple : Fonction factorielle : $n!$

- $\text{fac}(0) ::= 1$.
- $\text{fac}(n+1) ::= (n+1) \cdot \text{fac}(n)$ pour $n \geq 0$.

Autres exemples

Nombres de Fibonacci :

- $F_0 ::= 0$,
- $F_1 ::= 1$,
- $F_i ::= F_{i-1} + F_{i-2}$ pour $i \geq 2$.

Notation $\sum$:

- $\sum_{i=1}^{0} f(i) ::= 0$,
- $\sum_{i=1}^{n+1} ::= \sum_{i=1}^{n} f(i) + f(n+1)$, pour $n \geq 0$.

Définition croisée :

- $f(0) ::= 1$, $g(0) ::= 1$,
- $f(n+1) ::= f(n) + g(n)$, pour $n \geq 0$,
- $g(n+1) ::= f(n) \times g(n)$, pour $n \geq 0$

Définitions mal formées

Des problèmes peuvent apparaître lorsque la fonction ne suit pas la définition du type de données récursifs :

$$f_1(n) ::= \begin{cases} 0 & \text{si } n = 1, \\ f_1(n+1) & \text{sinon.} \end{cases}$$

(ne définit pas f_2 de manière unique)

$$f_2(n) ::= \begin{cases} 0 & \text{si } n \text{ est divisible par 2,} \\ 1, & \text{si } n \text{ est divisible par 3,} \\ 2, & \text{sinon.} \end{cases}$$

($f_3(6) = 0$ ou $f_3(6) = 1$)

$$f_3(n) ::= \begin{cases} 0 & \text{si } n = 0 \\ f_3(n+1) + 1, & \text{sinon.} \end{cases}$$

($f_3(1)$ n'est pas défini)

Trois fonctions bien définies

Suite de Syracuse :

$$f(n) ::= \begin{cases} 1 & \text{si } n \leq 1, \\ f(n/2) & \text{si } n > 1 \text{ est pair}, \\ f(3n+1) & \text{si } n > 1 \text{ est impair}. \end{cases}$$

Conjecture : f(n)=1 pour tout n. (Voir transparent 108)

Fonction d'Ackermann :

$$A(m,n) ::= \begin{cases} 2n, & \text{si } m = 0 \text{ ou } n \leq 1, \\ A(m-1, A(m, n-1)), & \text{sinon.} \end{cases}$$

Bien définie. Croissante extrêmement rapide : $A(4,4) \simeq 2^{2^{2^{65536}}}$

Fonction de McCarthy :

$$M(n) ::= \begin{cases} n-10, & \text{si } n > 100, \\ M(M(n+11)) & \text{si } n \leq 100. \end{cases}$$

$M(n) = 91$ pour tout $n \leq 101$, $n - 10$ sinon.

Expressions arithmétiques

Définition : Soit *Aexp* l'ensemble des expressions arithmétiques définies sur une seule variable x. *Aexp* peut être défini récursivement de la manière suivante :

- Cas de base :
  - x est dans *Aexp*.
  - $\forall k \in \mathbb{N}$, $\mathtt{k} \in Aexp$.[5]
- Cas récursifs : si $e, f \in Aexp$, alors :
  - $[e + f] \in Aexp$, (somme)
  - $[e * f] \in Aexp$, (produit)
  - $-[e] \in Aexp$. (négation)

Note : toutes les sous-expressions sont entre crochets et les exposants ne sont pas permis. $3x^2 + 2x + 1$ s'écrira dans *Aexp* :

$$[[3 * [x * x]] + [[2 * x] + 1]].$$

[5] $\mathtt{k}$ désigne une chaîne de caractère codant pour un nombre, et k désigne le nombre correspondant.

Evaluation d'une expression

On peut définir l'évaluation comme une fonction définie récursivement sur la structure de *Aexp*.

Définition : La fonction d'évaluation, eval : $Aexp \times \mathbb{Z} \to \mathbb{Z}$ est définie récursivement sur les expressions $e \in Aexp$ comme suit. Soit n un entier :

- Cas de base :
  - $\text{eval}(x, n) ::= n$,
  - $\text{eval}(\mathtt{k}, n) ::= k$.
- Cas récursifs :
  - $\text{eval}([e_1 + e_2], n) ::= \text{eval}(e_1, n) + \text{eval}(e_2, n)$,
  - $\text{eval}([e_1 * e_2], n) ::= \text{eval}(e_1, n) \cdot \text{eval}(e_2, n)$,
  - $\text{eval}(-[e_1], n) ::= -\text{eval}(e_1, n)$.

Exemple :

$$\begin{aligned} \text{eval}([3 + [x * x]], 2) &= \text{eval}(3, 2) + \text{eval}([x * x], 2) \\ &= 3 + \text{eval}([x * x], 2) \\ &= 3 + (\text{eval}(x, 2) \cdot \text{eval}(x, 2)) \\ &= 3 + (2 \cdot 2) \\ &= 7. \end{aligned}$$

Substitution

La fonction subst(f, e) remplace toutes les occurrences de x dans e par f.

Définition : La fonction subst : $Aexp \times Aexp \longrightarrow Aexp$ est définie récursivement sur les expressions $e \in Aexp$ comme suit. Soit $f \in Aexp$.

- Cas de base :
  - subst$(f, x) ::= f$,
  - subst$(f, \mathtt{k}) ::= \mathtt{k}$.
- Cas récursifs :
  - subst$(f, [e_1 + e_2]) ::= [\text{subst}(f, e_1) + \text{subst}(f, e_2)]$,
  - subst$(f, [e_1 * e_2]) ::= [\text{subst}(f, e_1) * \text{subst}(f, e_2)$,
  - subst$(f, -[e_1]) ::= -[\text{subst}(f, e_1)$.

Exercice : montrez que

$$subst([3 * x], [x * [x + -[1]]]) = [[3 * x] * [[3 * x] + -[1]]]$$

.

Modèle d'évaluation avec substitution

Supposons qu'on veuille évaluer l'expression $e' ::= \text{subst}(f, e)$ pour une valeur de $x = n$.

Deux possibilités :

- Calculer $\text{eval}(\text{subst}(f, e), n)$, c'est-à-dire remplacer x par f dans e et puis évaluer l'expression résultante pour $x = n$.
  (Modèle de substitution)
- Calculer $\text{eval}(e, \text{eval}(f, n))$, c'est-à-dire évaluer f pour $x = n$ et ensuite e pour x fixé à la valeur obtenue pour f.
  (Modèle d'environnement)

La deuxième approche est généralement plus efficace (d'autant plus que f est complexe et qu'il y a de x dans e).

Exercice : tester les deux approches pour $f = [3 * x]$, $e = [x * [x + -[1]]]$, $n = 2$.

Modèle d'évaluation avec substitution

En terme de langage de programmation :

- Modèle de substitution :

```
x=2;
return (3*x)*((3*x)+-(1))
```

  $\Rightarrow$ 3 multiplications

- Modèle d'environnement :

```
x=2
f=3*x
return f*(f+-(1))
```

  $\Rightarrow$ 2 multiplications

Equivalence des deux approches

Théorème : Pour toutes expressions $e, f \in \mathit{Aexp}$ et $n \in \mathbb{Z}$,

$$\text{eval}(\text{subst}(f, e), n) = \text{eval}(e, \text{eval}(f, n)).$$

Démonstration : La preuve se fait par induction structurelle sur e.

- Cas de base :
  - $e = x$: les membres de gauche et droite valent tous deux $\text{eval}(f, n)$.
  - $e = \texttt{k}$: par la définition de eval et subst, les deux membres valent $\texttt{k}$.
- Cas récursifs :
  - $e = [e_1 + e_2]$: Par hypothèse d'induction structurelle, on suppose que pour tout $f \in \mathit{Aexp}$, $n \in \mathbb{Z}$, et $i \in \{1, 2\}$

    $$\text{eval}(\text{subst}(f, e_i), n) = \text{eval}(e_i, \text{eval}(f, n)).$$

    Montrons que

    $$\text{eval}(\text{subst}(f, [e_1 + e_2]), n) = \text{eval}([e_1 + e_2], \text{eval}(f, n)).$$

Par la définition de subst, le membre de gauche vaut :

$$\text{eval}([\text{subst}(f, e_1) + \text{subst}(f, e_2)], n),$$

qui, par la définition de eval, est égal à :

$$\text{eval}(\text{subst}(f, e_1), n) + \text{eval}(\text{subst}(f, e_2), n).$$

Par hypothèse inductive, cette expression est égale à

$$\text{eval}(e_1, \text{eval}(f, n)) + \text{eval}(e_2, \text{eval}(f, n)).$$

Par définition de eval, cette dernière expression est égale au membre de droite, ce qui prouve le théorème dans ce cas.

- $e = [e_1 * e_2]$, $e = -[e_1]$ sont traités de manière similaire.

Tous les cas de base et récursifs étant traités, le théorème est prouvé par induction structurelle. □

Résumé

On a vu :

- comment définir des ensembles de manière récursive
- comment définir des fonctions sur les éléments d'un ensemble défini récursivement
- comment prouver l'appartenance d'un élément à un ensemble
- comment prouver des propriétés sur ces ensembles par le principe d'induction structurelle
- qu'il existe des définitions récursives ambiguës

Note : Dans tous nos exemples, on pourrait se passer de l'induction structurelle et s'en sortir avec l'induction ordinaire. Cependant :

- L'induction structurelle simplifie les preuves
- En théorie, l'induction structurelle est plus puissante que l'induction ordinaire

Applications

Les applications de ces principes sont innombrables en informatique :

- Définition de structure de données récursives (arbres, chaînes de caractère, listes, etc.)
- Calcul de fonctions et vérification de propriétés sur ces structures
- Définition de la syntaxe et de la sémantique de langages de programmation (voir le chapitre suivant)
- Fonctions récursives sur $\mathbb{N}$: dénombrement d'ensembles et analyse de complexité (voir la troisième partie du cours)