Construction d'analyseurs syntaxiques
TL2 Ensimag 1A 2022-2023

Chapitre 2
Exemple de la calculette

Xavier.Nicollin@grenoble-inp.fr

thanks Sylvain Boulmé *et* Lionel Rieg

Objectifs du chapitre

- Illustrer la notion d'interpréteur introduite au chapitre 1
- Spécifier la calculette que vous allez réaliser en TP
- Variation sur le projet de TL1...

Spécification informelle

- Entrée : séquence de *calculs* (expressions arithmétiques entières), chacun suivi de '?'
- Possibilité de faire référence au résultat d'un calcul antérieur via '#*n*' où *n* est le numéro du calcul
- Arrêt sur la première erreur (plus simple à implémenter)

Lexicographie

Rappel terminal de la BNF = token = langage régulier

Singletons $\texttt{PLUS} \stackrel{def}{=} \{\text{`+'}\}$ $\texttt{MINUS} \stackrel{def}{=} \{\text{`-'}\}$ $\texttt{MULT} \stackrel{def}{=} \{\text{`*'}\}$
$\texttt{DIV} \stackrel{def}{=} \{\text{`/'}\}$ $\texttt{QUEST} \stackrel{def}{=} \{\text{`?'}\}$ $\texttt{OPAR} \stackrel{def}{=} \{\text{`('}\}$ $\texttt{CPAR} \stackrel{def}{=} \{\text{`)'}\}$

Langages infinis $\texttt{NAT} \stackrel{def}{=} \{\text{`0'}, \ldots, \text{`9'}\}^+$ $\texttt{CALC} \stackrel{def}{=} \{\text{`\#'}\}.\texttt{NAT}$

Séparateurs '␣', tabulation, fin de ligne

Notations sur les listes dans la sémantique

But représenter "liste de résultats des calculs"

- Ensemble des listes d'entiers relatifs noté $\mathbb{L}$
- La taille d'une liste $\ell \in \mathbb{L}$ est un entier naturel noté $|\ell|$
- Liste vide notée $[\,]$ de taille $|[\,]| = 0$
- Si $\ell \in \mathbb{L}$ et $i \in [1, |\ell|]$, alors $\ell[i] \in \mathbb{Z}$ est le i-ème élément de ℓ
  NB "$\ell[i]$" opération partielle non-définie si $i \notin [1, |\ell|]$
- Si $\ell \in \mathbb{L}$ et $n \in \mathbb{Z}$, alors $\ell \oplus n \in \mathbb{L}$ est la liste obtenue en ajoutant n à la fin de ℓ
  On a $|\ell \oplus n| = |\ell| + 1$ et $(\ell \oplus n)[i] =$ si $i \leqslant |\ell|$ alors $\ell[i]$ sinon n

BNF attribuée

Profils input$\uparrow\mathbb{L}$ et exp$\downarrow\mathbb{L}\uparrow\mathbb{Z}$

NAT$\uparrow\mathbb{N}$ et CALC$\uparrow\mathbb{N}$ (retourne valeur de l'entier lu en base 10)

input$\uparrow\ell$	::=	ε	$\ell := []$
	|	input$\uparrow\ell_0$ exp$\downarrow\ell_0\uparrow n$ QUEST	$\ell := \ell_0 \oplus n$
exp$\downarrow\ell\uparrow n$	::=	NAT$\uparrow n$	
	|	CALC$\uparrow i$	$n := \ell[i]$
	|	exp$\downarrow\ell\uparrow n_1$ PLUS exp$\downarrow\ell\uparrow n_2$	$n := n_1 + n_2$
	|	exp$\downarrow\ell\uparrow n_1$ MINUS exp$\downarrow\ell\uparrow n_2$	$n := n_1 - n_2$
	|	exp$\downarrow\ell\uparrow n_1$ MULT exp$\downarrow\ell\uparrow n_2$	$n := n_1 \times n_2$
	|	exp$\downarrow\ell\uparrow n_1$ DIV exp$\downarrow\ell\uparrow n_2$	$n := n_1 \,/\, n_2$
	|	MINUS exp$\downarrow\ell\uparrow n_0$	$n := -n_0$
	|	OPAR exp$\downarrow\ell\uparrow n$ CPAR	

Table de priorités (*precedence*)

niveau 2 (priorité min)	associatif à gauche	`PLUS` binaire
		`MINUS` binaire
niveau 1	associatif à gauche	`MULT` binaire
		`DIV` binaire
niveau 0 (priorité max)		`MINUS` unaire

Exo 1 Donner arbre d'analyse de "`1+2*3?␣␣-4+#1*#1?`"
Donner la sémantique associée

Variation et remarques sur le projet TL1

Calculette en mode préfixe :

$$
\begin{array}{lcll}
\text{exp}{\uparrow}n & ::= & \texttt{NAT}{\uparrow}n & \\
 & | & \texttt{PLUS}\ \text{exp}{\uparrow}n_1\ \text{exp}{\uparrow}n_2 & n := n_1 + n_2 \\
 & | & \texttt{MINUS}\ \text{exp}{\uparrow}n_1\ \text{exp}{\uparrow}n_2 & n := n_1 - n_2 \\
 & \ldots & &
\end{array}
$$

```
def parse_exp():
    if current_token == NAT:
        n = val(current_token)
        advance_token
        return n
    elif current_token == PLUS:
        advance_token
        n_1 = parse_exp()
        n_2 = parse_exp()
        return n_1 + n_2
    elif current_token == MINUS:
        advance_token
        n_1 = parse_exp()
        n_2 = parse_exp()
        return n_1 - n_2
    # ...
```

Pourquoi est-ce facile ?

$$\begin{array}{lcll} \text{exp}\uparrow n & ::= & \texttt{NAT}\uparrow n & \\ & | & \texttt{PLUS}\ \text{exp}\uparrow n_1\ \text{exp}\uparrow n_2 & n := n_1 + n_2 \\ & | & \texttt{MINUS}\ \text{exp}\uparrow n_1\ \text{exp}\uparrow n_2 & n := n_1 - n_2 \\ & \ldots & & \end{array}$$

... car `parse_exp` décide de la règle selon `current_token`

```
def parse_exp():
    if current_token == NAT:
        n = val(current_token)
        advance_token
        return n
    elif current_token == PLUS:
        advance_token
        n_1 = parse_exp()
        n_2 = parse_exp()
        return n_1 + n_2
    # ...
```

Et en infixé c'est plus compliqué... suite du cours...