Le langage *Vier*

pour le cours de construction de compilateurs de 2007

révision γ_7

Grammaire lexicale

input	::=	{ whitespace | comment | token }
token	::=	*ident*
	|	*number*
	|	*string*
	|	“`true`” | “`false`”
	|	‘`(`’ | ‘`)`’ | ‘`{`’ | ‘`}`’
	|	‘`=`’ | ‘`:`’ | ‘`;`’ | ‘`.`’
	|	“`class`” | “`extends`”
	|	“`field`” | “`method`”
	|	“`Int`” | “`Null`”
	|	“`new`” | “`null`” | “`this`”
	|	“`if`” | “`then`” | “`else`”
	|	“`readInt`” | “`readChar`”
	|	“`printInt`” | “`printChar`”
	|	“`return`”
	|	‘`+`’ | ‘`-`’ | ‘`*`’ | ‘`/`’ | ‘`%`’
	|	“`==`” | ‘`<`’ | ‘`>`’ | “`&&`” | “`||`”
comment	::=	“`//`” { cchar }
ident	::=	letter { letter | digit | ‘`_`’ }
number	::=	‘`0`’ | digit1 { digit }
string	::=	‘`"`’ { schar } ‘`"`’
letter	::=	‘`a`’ | ⋯ | ‘`z`’ | ‘`A`’ | ⋯ | ‘`Z`’
digit	::=	‘`0`’ | digit1
digit1	::=	‘`1`’ | ⋯ | ‘`9`’

Un caractère est “whitespace” si la méthode “`Character.isWhitespace`” de la librairie Java l’accepte. Un caractère est “cchar” s’il ne définit pas de nouvelle ligne. Un caractère est “schar” s’il est “cchar” et pas ‘`"`’.

Dans la grammaire lexicale, “abc” doit être lu comme ‘a’ ‘b’ ‘c’.

Grammaire syntaxique

Program	::=	{ ClassDef } Expr
ClassDef	::=	“`class`” *ident* [“`extends`” *ident*] ‘`{`’ { Member } ‘`}`’
Member	::=	FieldDef
	|	MethDef
Type	::=	*ident* | “`Int`” | “`Null`”
Formal	::=	*ident* ‘`:`’ Type
Formals	::=	[Formal { ‘`;`’ Formal }]
FieldDef	::=	“`field`” Formal ‘`=`’ Expr
MethDef	::=	“`method`” *ident* ‘`(`’ Formals ‘`)`’ ‘`:`’ Type ‘`=`’ Expr
Expr	::=	“`if`” Expr “`then`” Expr “`else`” Expr
	|	CmpExpr
CmpOp	::=	“`==`” | ‘`<`’ | ‘`>`’
CmpExpr	::=	[SumExpr CompOp] SumExpr
SumOp	::=	‘`+`’ | ‘`-`’ | “`||`”
SumExpr	::=	[SumExpr SumOp] Term
ProdOp	::=	‘`*`’ | ‘`/`’ | ‘`%`’ | “`&&`”
Term	::=	[Term ProdOp] Factor
Factor	::=	*ident*
	|	*number*
	|	*string*
	|	“`true`” | “`false`”
	|	“`null`” | “`this`”
	|	“`new`” Type
	|	Factor ‘`.`’ *ident*
	|	Factor ‘`.`’ ‘`{`’ Assigns ‘`}`’
	|	Factor ‘`.`’ *ident* ‘`(`’ Exprs ‘`)`’
	|	‘`(`’ Expr ‘`)`’
	|	‘`{`’ Exprs “`return`” Expr ‘`}`’
	|	“`readInt`” | “`readChar`”
	|	“`printInt`” ‘`(`’ Expr ‘`)`’
	|	“`printChar`” ‘`(`’ Expr ‘`)`’
Exprs	::=	[Expr { ‘`;`’ Expr }]
Assigns	::=	*ident* ‘`=`’ Expr { ‘`;`’ *ident* ‘`=`’ Expr }

Sucre syntaxique

Syntaxe concrète	Syntaxe abstraite
`true`	1
`false`	0
e `||` f	$\neg(\neg e \wedge \neg e')$
$\neg e$	`if` $e = 0$ `then` 1 `else` 0
e `.{` a `=` f `}`	$e.a = f$
e `.{` a `=` f `;` `...` `}`	$(e.a = f)$ `.{` ... `}`
`""`	`new Nil`
`"a..."`	`(new Cons).{ head = a; tail = "..." }`

Grammaire abstraite

nom référençable			a, b	
nom de classe			t, u	
nombre	n			
programme	P	::=	$\overline{D}\ e$	
classe	D	::=	`class` $t \triangleleft s\ \{\, \overline{d}\,\}$	
	s	::=	t	
		|	`none`	
membre	d	::=	`field` $a : T\ =\ e$	
		|	`method` $a\ (\,\overline{a} : \overline{T}\,) : T\ =\ e$	
type	T, U	::=	t	
		|	`Int`	
		|	`Null`	
expression	e, f	::=	a	paramètre
		|	n	nombre
		|	`new` t	nouvelle instance
		|	`null`	valeur “`null`”
		|	$e.a$	sélection de champ
		|	$e.a(\overline{e})$	appel de méthode
		|	$e.a\ =\ e$	assignation de champ
		|	`if` e `then` e `else` e	condition “`if`”
		|	$e\ =\ e$	comparaison binaire
		|	$e\,p\,e$	op. binaire
		|	`readInt`	op. “`readInt`”
		|	`readChar`	op. “`readChar`”
		|	`printInt`(e)	op. “`printInt`”
		|	`printChar`(e)	op. “`printChar`”
		|	$\{\ \overline{e}$ `return` $e\ \}$	block
opérateur	p	::=	$+ \mid - \mid * \mid \div \mid \% \mid < \mid > \mid \wedge$	

Validité du typage

Symboles et portées

symbole de classe	σ_c	::=	$(\overline{t} \mid \Gamma_f \mid \Gamma_m)_c$
symbole de champ	σ_f	::=	$(T)_f$
symbole de méthode	σ_m	::=	$(\overline{T} \mid T)_m$
symbole de paramètre	σ_p	::=	$(T)_p$
portée des classes	Γ_c	::=	$\overline{t} \mapsto \overline{\sigma_c}$
portée des champs	Γ_f	::=	$\overline{a} \mapsto \overline{\sigma_f}$
portée des méthodes	Γ_m	::=	$\overline{a} \mapsto \overline{\sigma_m}$
portée des paramètres	Γ_p	::=	$\overline{a} \mapsto \overline{\sigma_p}$

Règles de typage

Validité d'un programme $\quad \overline{D}\ e \diamond$

$$\frac{\begin{array}{c}\texttt{none} \mapsto (\epsilon \,|\, \epsilon \,|\, \epsilon)_c \vdash \overline{D} \ \Rightarrow\ \Gamma'_c \\ \Gamma'_c \vdash \overline{D} \ \Rrightarrow\ \Gamma_c \qquad \Gamma_c \vdash \overline{D} \diamond \qquad \Gamma_c ; \epsilon \vdash e : T\end{array}}{\overline{D}\ e \diamond}$$

Validité de l'héritage entre classes $\quad \Gamma_c \vdash D \ \Rightarrow\ \Gamma'_c$

$$\frac{s \mapsto (\overline{s} \,|\, \epsilon \,|\, \epsilon)_c \in \Gamma_c \qquad \Gamma'_c = \Gamma_c \uplus t \mapsto (s, \overline{s} \,|\, \epsilon \,|\, \epsilon)_c}{\Gamma_c \vdash \texttt{class}\ t \ \lhd\ s\ \{\, \overline{d} \,\} \ \Rightarrow\ \Gamma'_c}$$

Validité de la déclaration des membres $\quad \Gamma_c \vdash D \ \Rrightarrow\ \Gamma'_c$

$$\frac{\begin{array}{c}s \mapsto (\overline{s} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \qquad \Gamma'_f = \Gamma_f + \mathit{fields}(\overline{d}) \\ \Gamma'_m = \Gamma_m + \mathit{methods}(\overline{d}) \qquad \Gamma'_c = \Gamma_c + t \mapsto (s, \overline{s} \,|\, \Gamma'_f \,|\, \Gamma'_m)_c\end{array}}{\Gamma_c \vdash \texttt{class}\ t \ \lhd\ s\ \{\, \overline{d} \,\} \ \Rrightarrow\ \Gamma'_c}$$

Validité d'une classe $\quad \Gamma_c \vdash D \diamond$

$$\frac{\Gamma_c ; \texttt{this} \mapsto (t)_p \vdash \overline{d} \diamond}{\Gamma_c \vdash \texttt{class}\ t \ \lhd\ s\ \{\, \overline{d} \,\} \diamond}$$

Validité d'un membre $\quad \Gamma_c ; \Gamma_p \vdash d \diamond$

$$\frac{\left.\begin{array}{r}\texttt{this} \mapsto (t)_p \in \Gamma_p \\ t \mapsto (s, \overline{s} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \\ s \mapsto (\overline{s} \,|\, \Gamma'_f \,|\, \Gamma'_m)_c \in \Gamma_c \\ a \mapsto (T')_f \in \Gamma'_f\end{array}\right\} \Rightarrow T = T' \qquad \Gamma_c \vdash T \diamond \qquad \Gamma_c ; \Gamma_p \vdash e : U \qquad \Gamma_c \vdash U \lhd T}{\Gamma_c ; \Gamma_p \vdash \texttt{field}\ a : T \ =\ e \diamond}$$

$$\frac{\left.\begin{array}{r}\texttt{this} \mapsto (t)_p \in \Gamma_p \\ t \mapsto (s, \overline{s} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \\ s \mapsto (\overline{s} \,|\, \Gamma'_f \,|\, \Gamma'_m)_c \in \Gamma_c \\ a \mapsto (\overline{T'} \,|\, T')_m \in \Gamma'_m\end{array}\right\} \Rightarrow \Gamma_c \vdash T \lhd T' \wedge \Gamma_c \vdash \overline{T'} \lhd \overline{T} \qquad \Gamma_c \vdash T \diamond \qquad \Gamma_c \vdash \overline{T} \diamond \qquad \Gamma_c ; \Gamma_p + \mathit{params}(\overline{a} : \overline{T}) \vdash e : U \qquad \Gamma_c \vdash U \lhd T}{\Gamma_c ; \Gamma_p \vdash \texttt{method}\ a\ (\, \overline{a} : \overline{T} \,) : T \ =\ e \diamond}$$

Validité d'une expression $\Gamma_c ; \Gamma_p \vdash e : T$

$$\frac{a \mapsto (T)_p \in \Gamma_p}{\Gamma_c ; \Gamma_p \vdash a : T} \qquad \Gamma_c ; \Gamma_p \vdash n : \texttt{Int} \qquad \frac{t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c}{\Gamma_c ; \Gamma_p \vdash \texttt{new}\ t : t}$$

$$\Gamma_c ; \Gamma_p \vdash \texttt{null} : \texttt{Null}$$

$$\frac{\Gamma_c ; \Gamma_p \vdash e : t \qquad t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \qquad a \mapsto (U)_f \in \Gamma_f}{\Gamma_c ; \Gamma_p \vdash e.a : U}$$

$$\frac{\begin{array}{c}\Gamma_c ; \Gamma_p \vdash e : t \qquad t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \\ a \mapsto (\overline{U} \,|\, U)_m \in \Gamma_m \qquad \Gamma_c ; \Gamma_p \vdash \overline{e} : \overline{T} \qquad \Gamma_c \vdash \overline{T} \triangleleft \overline{U}\end{array}}{\Gamma_c ; \Gamma_p \vdash e.a(\overline{e}) : U}$$

$$\frac{\begin{array}{c}\Gamma_c ; \Gamma_p \vdash e : t \qquad t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \\ a \mapsto (U)_f \in \Gamma_f \qquad \Gamma_c ; \Gamma_p \vdash f : T \qquad \Gamma_c \vdash T \triangleleft U\end{array}}{\Gamma_c ; \Gamma_p \vdash e.a \ = \ f : t}$$

$$\frac{\begin{array}{c}\Gamma_c ; \Gamma_p \vdash e : \texttt{Int} \\ \Gamma_c ; \Gamma_p \vdash f : U \qquad \Gamma_c ; \Gamma_p \vdash f' : U' \qquad \Gamma_c \vdash U \triangleleft T \triangleright U'\end{array}}{\Gamma_c ; \Gamma_p \vdash \texttt{if}\ e\ \texttt{then}\ f\ \texttt{else}\ f' : T}$$

$$\frac{\Gamma_c ; \Gamma_p \vdash e : T \qquad \Gamma_c ; \Gamma_p \vdash e' : T' \qquad \Gamma_c \vdash T \triangleleft T' \vee \Gamma_c \vdash T' \triangleleft T}{\Gamma_c ; \Gamma_p \vdash e \ = \ e' : \texttt{Int}}$$

$$\frac{\Gamma_c ; \Gamma_p \vdash e : \texttt{Int} \qquad \Gamma_c ; \Gamma_p \vdash e' : \texttt{Int}}{\Gamma_c ; \Gamma_p \vdash e \, p \, e' : \texttt{Int}} \qquad \Gamma_c ; \Gamma_p \vdash \texttt{readInt} : \texttt{Int}$$

$$\Gamma_c ; \Gamma_p \vdash \texttt{readChar} : \texttt{Int} \qquad \frac{\Gamma_c ; \Gamma_p \vdash e : \texttt{Int}}{\Gamma_c ; \Gamma_p \vdash \texttt{printInt}(e) : \texttt{Int}}$$

$$\frac{\Gamma_c ; \Gamma_p \vdash e : \texttt{Int}}{\Gamma_c ; \Gamma_p \vdash \texttt{printChar}(e) : \texttt{Int}} \qquad \frac{\Gamma_c ; \Gamma_p \vdash \overline{e} : \overline{U} \qquad \Gamma_c ; \Gamma_p \vdash f : T}{\Gamma_c ; \Gamma_p \vdash \{\ \overline{e}\ \texttt{return}\ f\ \} : T}$$

Validité d'un type $\Gamma_c \vdash T \diamond$

$$\frac{t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c}{\Gamma_c \vdash t \diamond} \qquad \Gamma_c \vdash \texttt{Null} \diamond \qquad \Gamma_c \vdash \texttt{Int} \diamond$$

Sous-typage $\Gamma_c \vdash T \triangleleft T$

$$\frac{t \mapsto (\overline{t} \,|\, \Gamma_f \,|\, \Gamma_m)_c \in \Gamma_c \qquad u \in \overline{t} \vee u = t}{\Gamma_c \vdash t \triangleleft u} \qquad \Gamma_c \vdash \texttt{Null} \triangleleft t$$

$$\Gamma_c \vdash \texttt{Null} \triangleleft \texttt{Null} \qquad \Gamma_c \vdash \texttt{Int} \triangleleft \texttt{Int}$$

Borne supérieure $\Gamma_c \vdash T \triangleleft T \triangleright T$

$$\frac{\Gamma_c \vdash T \triangleleft U \qquad \Gamma_c \vdash T' \triangleleft U \qquad \forall U' \,.\, \left.\begin{array}{c} \Gamma_c \vdash T \triangleleft U' \\ \Gamma_c \vdash T' \triangleleft U' \end{array}\right\} \Rightarrow \Gamma_c \vdash U \triangleleft U'}{\Gamma_c \vdash T \triangleleft U \triangleright T'}$$

Notations

Une notation définie sur x l'est pour tout élément, une notation définie sur Γ l'est pour toute portée.

Notation	Interprétation
$\overline{x}$	séquence $x_0, \ \ldots \ , x_n$ pour $n \in \mathbb{N}$
$x, \overline{x}$	séquence $x, x_0, \ \ldots \ , x_n$
$\overline{x} \mapsto \overline{\sigma}$	portée $x_0 \mapsto \sigma_0, \ldots, x_n \mapsto \sigma_n$ pour $n \in \mathbb{N}$
ϵ	portée ou séquence vide
$\Gamma_c \vdash \overline{D} \ \Rightarrow \ \Gamma'_c$	$\left\{ \begin{array}{l} \Gamma_c \vdash D_0 \ \Rightarrow \ \Gamma_c^0 \\ \vdots \\ \Gamma_c^{n-1} \vdash D_n \ \Rightarrow \ \Gamma'_c \end{array} \right.$
$\Gamma_c \vdash \overline{D} \ \Rrightarrow \ \Gamma'_c$	$\left\{ \begin{array}{l} \Gamma_c \vdash D_0 \ \Rrightarrow \ \Gamma_c^0 \\ \vdots \\ \Gamma_c^{n-1} \vdash D_n \ \Rrightarrow \ \Gamma'_c \end{array} \right.$
$\Gamma_c \vdash \overline{D} \diamond$	$\forall D \in \overline{D} \,.\, \Gamma_c \vdash D \diamond$
$\Gamma_c ; \Gamma_p \vdash \overline{d} \diamond$	$\forall d \in \overline{d} \,.\, \Gamma_c ; \Gamma_p \vdash d \diamond$
$\Gamma_c \vdash \overline{T} \diamond$	$\forall T \in \overline{T} \,.\, \Gamma_c \vdash T \diamond$
$\Gamma_c \vdash \overline{T} \triangleleft \overline{U}$	$\forall T, U \in \overline{T} \times \overline{U} \,.\, \Gamma_c \vdash T \triangleleft U$
$\Gamma_c ; \Gamma_p \vdash \overline{e} : \overline{T}$	$\forall e, T \in \overline{e} \times \overline{T} \,.\, \Gamma_c ; \Gamma_p \vdash e : T$
$\Gamma + x \mapsto \sigma$	$\left\{ \begin{array}{ll} \Gamma', x \mapsto \sigma, \Gamma'' & \text{si } \exists \Gamma', \Gamma'', \sigma' \,.\, \Gamma = (\Gamma', x \mapsto \sigma', \Gamma'') \\ \Gamma, x \mapsto \sigma & \text{sinon} \end{array} \right.$
$\Gamma \uplus x \mapsto \sigma$	$\Gamma, x \mapsto \sigma$ si $\forall \Gamma', \Gamma'', \sigma' \,.\, \Gamma \neq (\Gamma', x \mapsto \sigma', \Gamma'')$
$fields(\overline{d})$	$\biguplus_{\texttt{field}\ a{:}T = e \in \overline{d}} \ a \mapsto (T)_f$
$methods(\overline{d})$	$\biguplus_{\texttt{method}\ a\ (\overline{a}{:}\overline{T}){:}T = e \in \overline{d}} \ a \mapsto (\overline{T} \,\vert\, T)_m$
$params(\overline{a} : \overline{T})$	$\biguplus_{a:T \in \overline{a}:\overline{T}} \ a \mapsto (T)_p$

Etat des changements

révision γ_7 Les éléments du symbole de paramètre dans la règle de validité d'une méthode sont dans le bon ordre.

révision γ_6 La règle de sous-typage entre classes définit correctement une classe comme sous-type d'elle-même.

révision γ_5 Le sucre syntaxique pour la négation ($\neg e$) est correct.

révision γ_4 La définition de la borne supérieure est plus simple. Les règles de validité d'un membre sont correctes aussi lorsque le membre défini n'est pas hérité.

révision γ_3 La grammaire abstraite est syntaxiquement plus proche de la grammaire concrète. Le type "`Nothing`" s'appelle "`Null`".

révision γ_2 La liste des classes parentes du symbole d'une classe ne contient plus la classe elle-même. Le sous-typage entre "`Nothing`" et "`Nothing`" est correct.

révision γ_1 Ajoute les règles de validité du typage de *Vier*, et les notions annexes. Compactifie la notation de la grammaire abstraite et définit le sucre syntaxique.

révision β_4 Ajoute "`Nothing`".

révision β_3 Modifie la syntaxe d'assignement des valeurs à un champ, qui permet maintenant une forme abrégée qui assigne plusieurs champs simultanément.

révision β_2 Ajoute le paramètre manquant à "`printInt`" et "`printChar`" dans la grammaire abstraite et syntaxique.

révision β_1 Première version publique.