

INFO-F-403 : Language theory and compiling

Rapport projet partie 1

Simon Picard
Arnaud Rosette

October 23, 2014

1 Expressions régulières

Afin de décrire les mots (tokens) acceptés par le langage Iulius, nous utilisons les expressions régulières étendues.

Voici le tableau qui reprend l'ensemble des tokens du langage Iulius. Chaque ligne de celui-ci est composée du nom du token, le type dans l'enum LexicalUnit qui lui est associé et de l'expression régulière qui correspond à ce token.

Nom	Type	Expression régulière
Comment	/	#.*(\\r)?\\n
Circumflex (^)	POWER	\^
Percentage (%)	REMAINDER	%
Tilde (~)	BITWISE_NOT	~
Pipe ()	BITWISE_OR	\
Dollar (\$)	BITWISE_XOR	\\$
Dubble greater (»)	ARITHMETIC_SHIFT_RIGHT	>>
Dubble lower («)	ARITHMETIC_SHIFT_LEFT	<<
Dubble equal (==)	EQUALITY	==
Exclamation equal (!=)	INEQUALITY	!=
Function	FUNCTION	function
Return	RETURN	return
Arrow right (->)	MAP_TO	->
Question mark (?)	TERNARY_IF	\?
Exclamation mark (!)	NEGATION	!

Nom	Type	Expression régulière
Colon (:)	TERNARY_ELSE	:
Dubble ampersand (&&)	LAZY_AND	&&
Dubble pipe ()	LAZY_OR	\\
While	WHILE	while
For	FOR	for
Semicolon (;)	END_OF_INSTRUCTION	;
Println	PRINTLN	println
Const	CONST	const
Let	LET	let
Dubble colon (::)	TYPE_DEFINITION	::
Boolean (type)	BOOLEAN_TYPE	Bool
Real (type)	REAL_TYPE	FloatingPoint
Integer (type)	INTEGER_TYPE	Integer
Integer (cast)	INTEGER_CAST	int
Real (cast)	REAL_CAST	float
Read integer	READ_INTEGER	readint
Read real	READ_REAL	readfloat
Boolean (cast)	BOOLEAN_CAST	bool
Backslash (\)	INVERSE_DIVIDE	\\
Do	DO	do
End	END	end
Comma (,)	COMMA	,
Left parenthesis (()	LEFT_PARENTHESIS	\(
Right parenthesis ())	RIGHT_PARENTHESIS	\)
Minus sign (-)	MINUS	-
Plus sign (+)	PLUS	\+
Equal sign (=)	ASSIGNATION	=
Asterisk (*)	TIMES	*
Slash (/)	DIVIDE	/
True	BOOLEAN	true
False	BOOLEAN	false
Lower sign (<)	LESS_THAN	<
Greater sign (>)	GREATER_THAN	>
Lower or equals (<=)	LESS_OR_EQUALS_THAN	<=
Greater or equals (>=)	GREATER_OR_EQUALS_THAN	>=
If	IF	if
Else	ELSE	else
Elseif	ELSE_IF	elseif
Identifier	IDENTIFIER	([a-z] [A-Z] _)
Integer	INTEGER	([a-z] [A-Z] [0-9] _)*
Real	REAL	(([1-9] [0-9]*) 0) (([1-9] [0-9]*) 0) \. [0-9]+

2 Choix d'implémentation

2.1 Gestion des nombres et opérations

Afin de gérer les expressions arithmétiques contenant des nombres et les opérateurs plus et moins, nous avons dû supprimer les opérateurs plus et moins dans l'expression régulière des nombres entiers et réels. Par exemple, si on a $4+5$ on détectera "4", "+" et "5" avec notre implémentation tandis que si on avait laissé le plus dans l'expression régulière des entiers nous aurions obtenu "4" et "+5".

Par contre un entier $+2$ sera détecté comme "+" et "2" mais ce sera plus simple à interpréter lors de la construction de l'arbre syntaxique.

2.2 DFA

Premièrement, nous utilisons les notations suivantes :

- * $[a-z]$ signifie l'ensemble des lettres minuscules de a à z
- * $[A-Z]$ signifie l'ensemble des lettres majuscules de a à z
- * $[0-9]$ signifie l'ensemble des chiffres de 0 à 9
- * Par extension $[d-y]$ signifie l'ensemble des lettres minuscules de d à y
- * $[.]$ signifie le . dans les expressions régulières soit tous les caractères

Ensuite, dans la partie à droite de l'état initiale dans le graphe, soit les différents mots-clés et les identifiants, pour plus de lisibilité nous n'avons pas inclus les transitions d'un état faisant partie d'un mot-clé vers un identifiant, normalement chaque état d'un mot-clé devrait contenir une transition depuis lui même vers l'état identifier, la transition comprend $\{[a-z], [A-Z], [0-9], _ \}$ en excluant les autres transitions sortantes de cet état.