Teoría de Lenguajes

Trabajo Práctico 1

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Formateador de Código Jay

Integrante	LU	Correo electrónico
Aronson, Alex	443/08	alexaronson@gmail.com
Alvarez, Daniel		salvarez@dc.uba.ar
Nahabedian, Leandro	250/08	leanahabedian@hotmail.com
Ravasi, Nicolás	53/08	nravasi@gmail.com

Contents

1 Resolución

Analizando las gramaticas del lenguaje Jay dada por la catedra y el funcionamiento de las herramientas propuestas, pudimos observar que la gramatica no era $LL(1)$ debido a que tenía recursión a izquierda principalmente. Lo que decidimos hacer entonces es utilizar la herramienta *JavaCC* que si bien solo funciona para gramaticas ELL con un pequeño pasaje donde agregaremos los simbolos *, $^+$ y ?, podremos obtener facilmente la gramatica ELL deseada.

1.1 Transformación de la gramática del lexer

Dados los siguientes componentes léxicos de Jay:

$InputElement \rightarrow WhiteSpace \mid Comment \mid Token$
$WhiteSpace \rightarrow$ **space** | \ t | \r | \n | \f | \r\n
$Comment \rightarrow //$ any string ended by \r or \n or \r\n
$Token \rightarrow Identifier \mid Keyword \mid Literal \mid Separator \mid Operator$
$Identifier \rightarrow Letter \mid IdentifierLetter \mid IdentifierDigit$
$Letter \rightarrow$ a | b | ... | z | A | B | ... | Z
$Digit \rightarrow$ 0 | 1 | 2 | ... | 9
$Keyword \rightarrow$ **boolean** | **else** | **if** | **int** | **main** | **void** | **while**
$Literal \rightarrow Boolean \mid Integer$
$Boolean \rightarrow$ **true** | **false**
$Integer \rightarrow Digit \mid IntegerDigit$
$Separator \rightarrow$ **(**|**)** | **{**|**}** | **;** | **,**
$Operator \rightarrow = \mid + \mid$ **-** | ***** | / | < | <= | > |
|>= | == | **!=** | **&&** | || | **!**

Los reescribiremos de la siguiente manera:

$InputElement \rightarrow WhiteSpace \mid Comment \mid Token$
$WhiteSpace \rightarrow$ **space** | \ t | \r | \n | \f | \r\n
$Comment \rightarrow //.*\$$
$Token \rightarrow Identifier \mid Keyword \mid Literal \mid Separator \mid Operator$
$Identifier \rightarrow Letter \mid IdentifierLetter \mid IdentifierDigit$
$Letter \rightarrow$ [a-zA-Z]
$Digit \rightarrow$ [0-9]
$Keyword \rightarrow$ **boolean** | **else** | **if** | **int** | **main** | **void** | **while**
$Literal \rightarrow Boolean \mid Integer$
$Boolean \rightarrow$ **true** | **false**
$Integer \rightarrow Digit \mid IntegerDigit$
$Separator \rightarrow$ **(**|**)** | **{**|**}** | **;** | **,**
$Operator \rightarrow = \mid + \mid$ **-** | ***** | / | < | <= | > |
| >= | == | **!=** | **&&** | || | **!**

Luego eliminamos las recursiones usando * o $^+$ y factorizamos utilizando ? entonces Integer va a ser:

$Integer \rightarrow [0-9]^+$
$Identifier \rightarrow [a-zA-Z][a-zA-Z0-9]^*$
$Operator \rightarrow =(=)? \mid + \mid$ **-** | ***** | / | <(=)? |

$| >(=)? \mid \textbf{!}(=)? \mid \textbf{\&\&} \mid \textbf{||}$

Quedando la siguiente gramatica

$InputElement \rightarrow WhiteSpace \mid Comment \mid Token$
$WhiteSpace \rightarrow \textbf{space} \mid \backslash\ t \mid \backslash r \mid \backslash n \mid \backslash f \mid \backslash r\backslash n$
$Comment \rightarrow //.*\$$
$Token \rightarrow Identifier \mid Keyword \mid Literal \mid Separator \mid Operator$
$Identifier \rightarrow [a-zA-Z]([a-zA-Z0-9])^*$
$Keyword \rightarrow \textbf{boolean} \mid \textbf{else} \mid \textbf{if} \mid \textbf{int} \mid \textbf{main} \mid \textbf{void} \mid \textbf{while}$
$Literal \rightarrow Boolean \mid Integer$
$Boolean \rightarrow \textbf{true} \mid \textbf{false}$
$Integer \rightarrow [0-9]^+$
$Separator \rightarrow \textbf{(|)} \mid \textbf{\{|\}} \mid \textbf{;} \mid \textbf{,}$
$Operator \rightarrow =(=)? \mid + \mid \textbf{-} \mid \textbf{*} \mid / \mid <(=)? \mid$
$\qquad | >(=)? \mid \textbf{!}(=)? \mid \textbf{\&\&} \mid \textbf{||}$

1.2 Transformación de la gramatica de la sintaxis de Jay

La idea ahora es realizar la misma acción para la gramática que define la sintaxis de Jay:

$Program \rightarrow \textbf{void main ( ) \{}Declarations\ Statements\textbf{\}}$
$Declarations \rightarrow \lambda \mid Declarations\ Declaration$
$Declaration \rightarrow Type\ Indentifiers\ \textbf{;}$
$Type \rightarrow \textbf{int} \mid \textbf{boolean}$
$Identifiers \rightarrow Identifier \mid Identifiers\textbf{,}Identifier$
$Statements \rightarrow \lambda \mid Statements\ Statement$
$Statement \rightarrow \textbf{;} \mid Block \mid Assignment \mid IfStatement \mid WhileStatement$
$Block \rightarrow \textbf{\{}Statements\textbf{\}}$
$Assignment \rightarrow Identifier = Expression\ \textbf{;}$
$IfStatement \rightarrow \textbf{if(}\ Expression\ \textbf{)}\ Statement \mid \textbf{if(}\ Expression\ \textbf{)}\ Statement\ \textbf{else}\ Statement$
$WhileStatement \rightarrow \textbf{while(}Expression\ \textbf{)}\ Statement$
$Expression \rightarrow Conjuntion \mid Expression\ \textbf{||}\ Conjunction$
$Conjunction \rightarrow Relation \mid Conjunction\ \&\&\ Relation$
$Relation \rightarrow Addition \mid$
$\qquad |Relation < Addition \mid Relation <= Addition \mid$
$\qquad |Relation > Addition \mid Relation >= Addition \mid$
$\qquad |Relation == Addition \mid Relation\ \textbf{!}= Addition$
$Addition \rightarrow Term \mid Addition + Term \mid Addition\ \textbf{-}\ Term$
$Term \rightarrow Negation \mid Term\ \textbf{*}\ Negation \mid Term\ /\ Negation$
$Negation \rightarrow Factor \mid \textbf{!}\ Factor$
$Factor \rightarrow Identifier \mid Literal \mid \textbf{(}\ Expression\ \textbf{)}$

Ahora vamos a eliminar las recursiones a izquierda como las de la producion *Declarations*, *Identifiers*, *Statements*, *Expression*, *Conjunction*, *Relation*, *Addition*, *Term*:

$Declarations \rightarrow (Declaration)^*$
$Identifiers \rightarrow Identifier\ (\textbf{,}Identifier)^*$

$Statements \rightarrow Statement^*$
$Expression \rightarrow Conjuntion\ (||\ Conjunction)^*$
$Conjunction \rightarrow Relation\ (\&\&\ Relation)^*$
$Relation \rightarrow Addition\ (Comparison\ Addition)^*$
$Comparison \rightarrow < \mid <= \mid > \mid >= \mid == \mid \mathbf{!=}$
$Addition \rightarrow Term\ (Sums\ Term)^*$
$Sums \rightarrow + \mid \mathbf{-}$
$Term \rightarrow Negation\ (Mult\ Negation)^*$
$Mult \rightarrow \mathbf{*} \mid /$

Note que para poder extender la producción *Relation*, *Addition* y $Term$ tuvimos que agregar tres producciones auxiliares, *Comparison*, *Sums* y $Mult$ respectivamente. Además como *Declarations* y *Statements* llaman a otra produccion ninguna o muchas veces, vamos a eliminar estas dos producciones y colocar en su reemplazo el lado derecho del mismo. Pasemos ahora a colocar los ? que es otro caracter que extiende la gramatica original y me va a reducir un poco mas lo que ya tengo. Agregaremos este símbolo en las siguientes producciones:

$IfStatement \rightarrow \mathbf{if(}\ Expression\ \mathbf{)}\ Statement\ (\mathbf{else}\ Statement)?$
$Negation \rightarrow (\mathbf{!})?\ Factor$
$Comparison \rightarrow <(=)? \mid >(=)? \mid == \mid \mathbf{!=}$

Veamos como quedan todos estos cambios al ponerlos todos juntos

$Program \rightarrow \mathbf{void\ main\ (\ )\ \{}Declaration^*\ Statement^*\mathbf{\}}$
$Declaration \rightarrow Type\ Indentifiers\ \mathbf{;}$
$Type \rightarrow \mathbf{int} \mid \mathbf{boolean}$
$Identifiers \rightarrow Identifier\ (\mathbf{,}Identifier)^*$
$Statement \rightarrow \mathbf{;} \mid Block \mid Assignment \mid IfStatement \mid WhileStatement$
$Block \rightarrow \mathbf{\{}Statement^*\mathbf{\}}$
$Assignment \rightarrow Identifier = Expression\ \mathbf{;}$
$IfStatement \rightarrow \mathbf{if(}\ Expression\ \mathbf{)}\ Statement\ (\mathbf{else}\ Statement)?$
$WhileStatement \rightarrow \mathbf{while(}Expression\ \mathbf{)}\ Statement$
$Expression \rightarrow Conjuntion\ (||\ Conjunction)^*$
$Conjunction \rightarrow Relation\ (\&\&\ Relation)^*$
$Relation \rightarrow Addition\ (Comparison\ Addition)^*$
$Comparison \rightarrow <(=)? \mid >(=)? \mid == \mid \mathbf{!=}$
$Addition \rightarrow Term\ (Sums\ Term)^*$
$Sums \rightarrow + \mid \mathbf{-}$
$Term \rightarrow Negation\ (Mult\ Negation)^*$
$Mult \rightarrow \mathbf{*} \mid /$
$Negation \rightarrow (\mathbf{!})?\ Factor$
$Factor \rightarrow Identifier \mid Literal \mid \mathbf{(}\ Expression\ \mathbf{)}$

Lo que nosotros elegimos como herramienta para hacer el trabajo práctico fue usar Javacc en el lenguaje Java. Al tener esta gramática definida pasamos a definir un archivo Jay.jj que se encargará de leer esta gramatica y darnos el parser deseado.

1.3 Traducción dirigida por sintaxis

Ahora a la gramática vamos a tener que agregarle atributos para obtener luego de parsear todo el código Jay un código HTML.

Para mas detalle de esto, ver el código en Jay.jj.

2 Referencias

1) javacc.java.net