Etapa 2

Analizador Sintáctico

Compiladores e Intérpretes

Entrega: 9 de Septiembre

Integrantes: **Brenda Soledad Dilschneider. LU: 92774**

Cuenca Francisco. LU:94294

Gramática Minijava

Eliminación de los elementos de la gramática EBNF

<Inicial> → <Clase> <Inicial> | <Clase>
<Clase> → class identificador <Herencia> { <MiembroL>} | class identificador {<miembroL>}
<MiembroL> → <Miembro> <MiembroL> | ƛ
<Herencia> → extends identificador
<Miembro> → <Atributo> | <Ctor> | <Metodo>
<Atributo> → varinst <Tipo> <ListaDecVars>;
<Metodo> → <Modmetodo> <TipoMetodo> identificador <ArgsFormales> <Bloque>
<Ctor> → identificador <ArgsFormales> <Bloque>
<ArgsFormales> → (<ListaArgsFormales>) | ()
<ListaArgsFormales> → <ArgFormal> | <ArgFormal> , <ListaArgsFormales>
<ArgFormal> → <Tipo> identificador
<ModMetodo> → static | dynamic
<TipoMetodo> → <Tipo> | void
<Tipo> → <TipoPrimitivo> | identificador
<TipoPrimitivo> → boolean | char | int | string
<ListaDecVars> → identificador | identificador , <ListaDecVars>
<Bloque> → {<SentenciaL>}
<SentenciaL> → <Sentencia> <SentenciaL> | ƛ
<Sentencia> → ;
<Sentencia> → <Asignación>;
<Sentencia> → <SentenciaSimple>;
<Sentencia> → varlocal <Tipo> <ListaDecVars>;
<Sentencia> → if (<Expresion>) <Sentencia>
<Sentencia> → if (<Expresion>) <Sentencia> else <Sentencia>
<Sentencia> → while (<Expresion>) <Sentencia>
<Sentencia> → for (<Asignación> ; <Expresión>; <Asignación>) | <Sentencia>
<Sentencia> → <Bloque>
<Sentencia> → return <Expresion> ; | return ;
<Asignacion> → <LadoIzquierdo> = <Expresion>
<LadoIzquierdo> → identificador | identificador . <LadoIzquierdo>
<SentenciaSimple> → (<Expresion>)
<Expresion> → <Expresion> || <Expr5> | <Expr5>
<Expr5> → <Expr5> && <Expr4> | <Expr4>
<Expr4> → <Expr4> == <Expr3> | < Expr4> != <Expr3> | <Expr3>
<Expr3> → <Expr3> >= <Expr2> | <Expr3> <= <Expr2> | <Expr3> < <Expr2> | <Expr3> > <Expr2> |
<Expr2>
<Expr2> → <Expr2 > - <Expr1> | <Expr2> + <Expr1> | <Expr1>
<Expr1> → <Expr1> * <ExprUnaria> | <Expr1> / <ExprUnaria> | <Expr1> % <ExprUnaria> | <ExprUnaria>
<ExprUnaria> → + <ExprUnaria> | - <ExprUnaria> | ! <ExprUnaria> | <Primario>
<Primario> → this
<Primario> → <Literal>

<Primario> → (<Expresion>) <LlamadaoIdEnc>
<Primario> → identificador<LlamadaoIdEnc>
<LlamadaoIdEnc> → <Llamada> <LlamadaoIdEnc> | ƛ
<Llamada> → . identificador <ArgsActuales> | . identificador
<Literal> → null | true | false | intLiteral | charLiteral | stringLiteral
<ArgsActuales> → (<ListaExps>) | ()
<ListaExps> → <Expresion> | <Expresion> , <ListaExps>

Eliminación de recursión a izquierda

<Inicial> → <Clase> <Inicial> | <Clase>
<Clase> → class identificador <Herencia> { <MiembroL>} | class identificador {<miembroL>}
<MiembroL> → <Miembro> <MiembroL> | ƛ
<Herencia> → extends identificador
<Miembro> → <Atributo> | <Ctor> | <Metodo>
<Atributo> → varinst <Tipo> <ListaDecVars>;
<Metodo> → <Modmetodo> <TipoMetodo> identificador <ArgsFormales> <Bloque>
<Ctor> → identificador <ArgsFormales> <Bloque>
<ArgsFormales> → (<ListaArgsFormales>) | ()
<ListaArgsFormales> → <ArgFormal> | <ArgFormal> , <ListaArgsFormales>
<ArgFormal> → <Tipo> identificador
<ModMetodo> → static | dynamic
<TipoMetodo> → <Tipo> | void
<Tipo> → <TipoPrimitivo> | identificador
<TipoPrimitivo> → boolean | char | int | string
<ListaDecVars> → identificador | identificador , <ListaDecVars>
<Bloque> → {<SentenciaL>}
<SentenciaL> → <Sentencia> <SentenciaL> | ƛ
<Sentencia> → ;
<Sentencia> → <Asignación>;
<Sentencia> → <SentenciaSimple>;
<Sentencia> → varlocal <Tipo> <ListaDecVars>;
<Sentencia> → if (<Expresion>) <Sentencia>
<Sentencia> → if (<Expresion>) <Sentencia> else <Sentencia>
<Sentencia> → while (<Expresion>) <Sentencia>
<Sentencia> → for (<Asignación> ; <Expresión>; <Asignación>) | <Sentencia>
<Sentencia> → <Bloque>
<Sentencia> → return <Expresion> ; | return ;
<Asignacion>→<LadoIzquierdo> = <Expresion>
<LadoIzquierdo> → identificador | identificador . <LadoIzquierdo>
<SentenciaSimple> → (<Expresion>)
<Expresion> → <Expr5> <ExprP>
<ExprP> → || <Expr5> <ExprP> | ƛ
<Expr5> → <Expr4> <Expr5P>
<Expr5P> → && <Expr4><Expr5P> | ƛ
<Expr4> → <Expr3> <Expr4P>

<Expr4P> → == <Expr3> <Expr4P> | **!**= <Expr3> <Expr4P> | ƛ
<Expr3> → <Expr2> >= <Expr2> | <Expr2> <= <Expr2> | <Expr2> < <Expr2> | <Expr2> > <Expr2> |
<Expr2>
<Expr2> → <Expr1> <Expr2P>
<Expr2P> → - <Expr1> <Expr2P> | + <Expr1> <Expr2P> | ƛ
<Expr1> → <ExprUnaria> <Expr1P>
<Expr1P> → *<ExprUnaria> <Expr1P> | /<ExprUnaria> <Expr1P> | %<ExprUnaria> <Expr1P>| ƛ
<ExprUnaria> → + <ExprUnaria> | - <ExprUnaria> | ! <ExprUnaria> | <Primario>
<Primario> → **this**
<Primario> → <Literal>
<Primario> → **(** <Expresion> **)** <LlamadaL>
<Primario> → identificador <LlamadaL>
<Primario> → **new** identificador <ArgsActuales> <LlamadaL>
<Primario> → identificador <ArgsActuales> <LlamadaL> <LlamadaL> → <Llamada><LlamadaL> | ƛ
<Llamada> → **.**identificador <ArgsActuales>
<Literal> → null | true | false | intLiteral | charLiteral | stringLiteral
<ArgsActuales> → **(** <ListaExps> **)** | **()**
<ListaExps> → <Expresion> | <Expresion> **,** <ListaExps>

Factorización

<Inicial> → <Clase> <InicialP>
<InicialP> → <Inicial> | ƛ
<Clase> → class identificador <ClaseP>
<ClaseP> → <Herencia> { <MiembroL>} | {<MiembroL>}
<MiembroL> → <Miembro> <MiembroL> | ƛ
<Herencia> → extends identificador
<Mmiembro> → <Atributo>| <Ctor>| <Metodo>
<Atributo> → varInst <Tipo> <ListaDecVars> ;
<Metodo> → <ModMetodo> <TipoMetodo> identificador <ArgsFormales> <Bloque>
<Ctor> → identificador <ArgsFormales> <Bloques>
<ArgsFormales> → (<ArgsFormalesP>
<ArgsFormalesP> → <ListaArgsFormales>) |)
<ListaArgsFormales> → <ArgFormal> <ListaArgsFormalesP>
<ListaArgsFormalesP> → , <ListaArgsFormales> | ƛ
<ArgFormal> → <Tipo> identificador
<ModMetodo> → static | dynamic
<TipoMetodo> → <Tipo> | void
<Tipo> → <TipoPrimitivo> | identificador
<TipoPrimitivo> → boolean | char | int | String
<ListaDecVars> → identificador <ListaDecVarsP>
<ListaDecVarsP> → , <ListaDecVars> | ƛ
<Bloque> → {<SentenciaL>}
<SentenciaL> → <Sentencia> <SentenciaL> | ƛ
<Sentencia> → ;

<Sentencia> → <Asignacion>
<Sentencia> → <SentenciaSimple>;
<Sentencia> → varLocal <Tipo><ListaDecVars>;
<Sentencia> → if (<Expresion>) <Sentencia> <SentenciaP>
<SentenciaP> → else <Sentencia> | ƛ
<Sentencia> → while (<Expresion>) <Sentencia>
<Sentencia> → for (<Asignacion> ; <Expresion> ; <Asignacion>) <Sentencia>
<Sentencia> → <Bloque>
<Sentencia> → return <SentenciaPP>
<SentenciaPP> → <Expresion> ; | ;
<Asignacion> → <LadoIzquierdo> = <Expresion>
<LadoIzquierdo> → identificador | identificador . <LadoIzquierdo>
<SentenciaSimple> → (<Expresion>)
<Expresion> → <Expr5> <ExprP>
<ExprP> → || <Expr5> <ExprP> | ƛ
<Expr5> → <Expr4> <Expr5P>
<Expr5P> → && <Expr4> <Expr5P> | ƛ
<Expr4> → <Expr3> <Expr4P>
<Expr4P> → == <Expr3> <Expr4P> | != <Expr3> <Expr4P> | ƛ
<Expr3> → <Expr2> <Expr3P>
<Expr3P> → >= <Expr2> || <= <Expr2> || > <Expr2> || < <Expr2> || ƛ
<Expr2> → <Expr1> <Expr2P>
<Expr2P> → - <Expr1> <Expr2P> | + <Expr1> <Expr2P> | ƛ
<Expr1> → <ExprUnaria> <Expr1P>
<Expr1P> → * <ExprUnaria> <Expr1P> | / <ExprUnaria> <Expr1P> | % <ExprUnaria> <Expr1P> | ƛ
<ExprUnaria> → + <ExprUnaria> | - <ExprUnaria> | ! <ExprUnaria> | <Primario>
<Primario> → this
<Primario> → <Literal>
<Primario> → (<Expresion>) <LlamadaL>
<Primario> → new identificador <ArgsActuales> <LlamadaL>
<Primario> → identificador <PrimarioP>
<PrimarioP> → <LlamadaL> | <ArgsActuales> <LlamadaL>
<LlamadaL> → <Llamada> <LlamadaL> | ƛ
<Llamada> → . identificador <ArgsActuales>
<Literal> → null | true | false | intLiteral | charLiteral | stringLiteral
<ArgsActuales> → (<ArgsActualesP>
<ArgsActualesP> → <ListaExps>) |)
<ListaExps> → <Expresion> <ListaExpsP>
<ListaExpsP> → , <ListaExps> | ƛ

Ambigüedad de la gramática

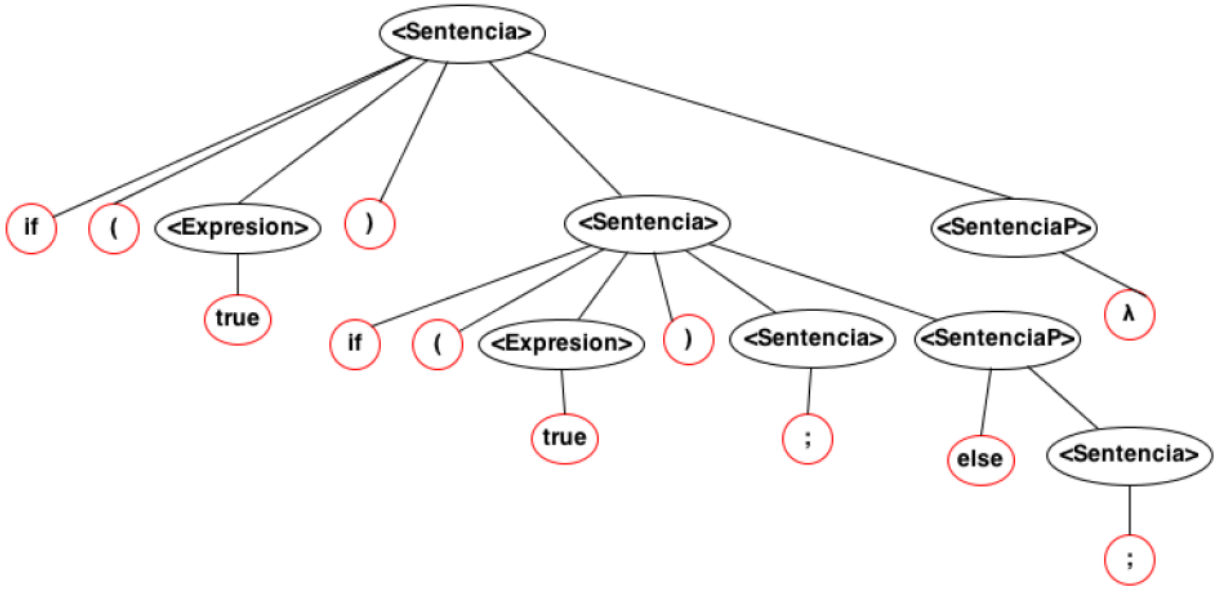
La gramática obtenida como resultado de las transformaciones de factorización y eliminación de recursividad a izquierda, no es de tipo LL(1) ya que es ambigua. La ambigüedad se da en las siguientes reglas de producción:

<Sentencia> → if (<Expresion>) <Sentencia> <SentenciaP>

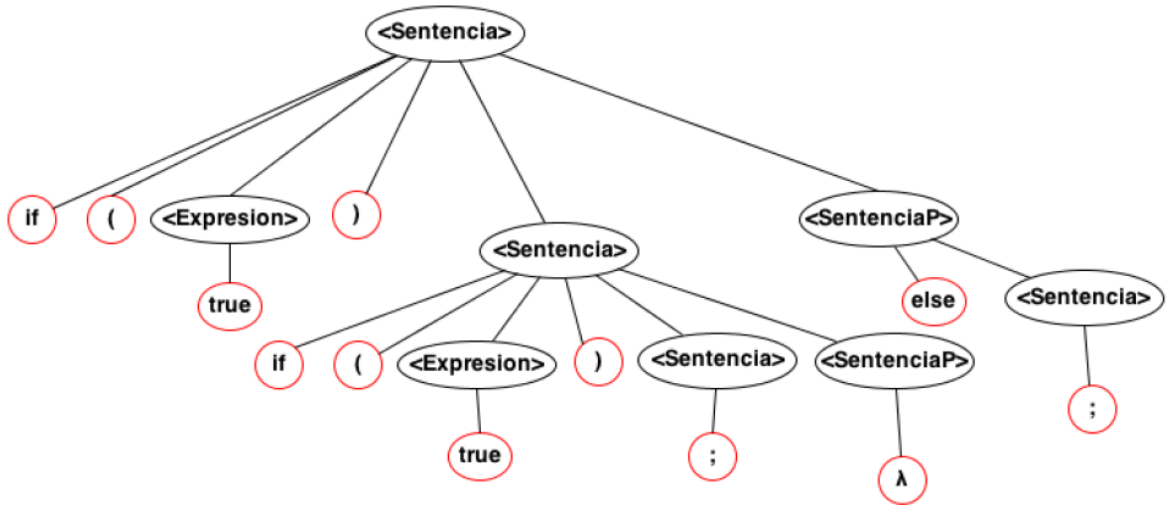
<SentenciaP> → else <Sentencia> | ƛ

Justificación:
Para la cadena “if (true) if (true); else ; “ que representa una sentencia, es posible obtener dos árboles de derivación distintos. El problema es que no se especifica en la gramática a que “if” corresponde el primer “else” encontrado.

1) Primer árbol de derivación:



2) Segundo árbol de derivación:



Solución en la implementación

En la implementación se soluciona la ambigüedad de forma tal que siempre se corresponde con el árbol de derivación (1). Esto es que el primer “else” siempre se corresponde con el “if” más cercano.

Errores detectados

El analizador sintáctico desarrollado es capaz de detectar todos los errores sintácticos. Al producirse dicho error, se arroja una excepción de tipo “SintaxError”, y el mismo indica al usuario en que línea se produjo el error, que se esperaba y que se encontró.

Solución en la implementación

En la implementación se soluciona la ambigüedad de forma tal que siempre se corresponde con el árbol de derivación (1). Esto es que el primer “else” siempre se corresponde con el “if” más cercano.

Errores detectados

El analizador sintáctico desarrollado es capaz de detectar todos los errores sintácticos. Al producirse dicho error, se arroja una excepción de tipo “SintaxError”, y el mismo indica al usuario en que línea se produjo el error, que se esperaba y que se encontró.