Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Proyecto Nº 2

Aplicación utilizando JLex y CUP (versión 1.1)

Segundo Cuatrimestre de 2010

La realización del presente proyecto, tiene como principal interés la experimentación con herramientas automáticas de desarrollo de compiladores, en este caso particular con JLex y CUP.

El trabajo cosistirá en el desarrollo de un pequeño intérprete. Para tal fin se les brindará un lenguaje particular muy sencillo.

1. Especificación del Mini-Lenguaje

Considérese un lenguaje que es un subconjunto de Standard ML. El objetivo del proyecto es construir un intérprete que a partir de las ligaduras y declaraciones especificadas, infiera y chequee los tipos de las entidades declaradas. **El intérprete no realizará la evaluación ni la ejecución del código especificado.** Simplemente se limitará a mostrar por pantalla las ligaduras producidas indicando el tipo adquirido por la entidad declarada/ligada.

El subconjunto del lenguaje ML a considerar admite expresiones aritméticas, booleanas y relacionales sobre los tipos básicos `bool`, `int`, `real` (sólo números en notación de punto fijo) y `string`. Como tipos estructurados sólo se tendrán tuplas (producto cartesiano), listas (secuencia) y funciones. También se incluirá polimorfismo paramétrico mediante el uso de politipos ($'a$) y de politipos con igualdad ($''a$). A nivel de sentencias, sólo se admitirán ligaduras explícitas (mediante `val`), implícitas (usando la variable `it`) y funcionales (declaración de funciones mediante `fun`).

En lineas generales el esquema del lenguaje es el siguiente:

<Programa> → <Lista_sentencias> **.**

<Lista_sentencias> → <Lista_sentencias> <Sentencia> **;** | <Sentencia> **;**

<Sentencia> → **val id** = <Exp> | <Exp> | **fun** <MatchingFunc>

<MatchingFunc> → **id id** <CurryingList> | **id** <TuplePattern> = <Exp>

<CurryingList> → **id** <CurryingList> | = <Exp>

<TuplePattern> → **(** **id** <TuplePattern2> **)**

<TuplePattern2> → **,** **id** <TuplePattern2> | λ

<Exp> → <Exp> + <Exp> | <Exp> **-** <Exp> | <Exp> ***** <Exp> |
<Exp> **/** <Exp> | <Exp> **div** <Exp> | <Exp> **mod** <Exp> |
<Exp> **<** <Exp> | <Exp> **>** <Exp> | <Exp> **<=** <Exp> |
<Exp> **>=** <Exp> | <Exp> **<>** <Exp> | <Exp> **=** <Exp> |
<Exp> **andalso** <Exp> | <Exp> **orelse** <Exp> | **not** <Exp> |
<Exp> **^** <Exp> | <Exp> <Exp> | ~ <Exp> |
int | **real** | **string** | **id** | <Exp> **::** <Exp> | **nil** |
(<Exp> **)** | <Tuple> | <List>

<Tuple> → **(** <Exp> <Tuple2> **)**

<Tuple2> → **,** <Exp> <Tuple2> | λ

<List> → [<Exp> <List2>]

<List2> → **,** <Exp> <List2> | λ

Los operadores de menor precedencia en el Mini-Lenguaje son los relacionales. Incrementando la precedencia, en el siguiente nivel se tiene al operador cons (**::**). Un nivel más arriba están los operadores aditivos (+, **-**, **orelse**) y la concatenación de strings (^). En el siguiente nivel están los operadores multiplicativos (*, /, **div**, **mod** y **andalso**). En el nivel que sigue se ubican los operadores 1-arios (∼, **not**) y, finalmente, en el nivel de mayor precedencia tenemos la aplicación funcional. La asociatividad de todos los operadores es a izquierda salvo para los operadores relacionales (que son no asociativos) y para el operador cons (el cual asocia derecha). También es importante destacar que, a diferencia de Standard ML, **el Mini-Lenguaje propuesto no es case sensitive.**

El intérprete deberá evaluar cada sentencia introducida por el usuario, inferir los tipos de las entidades ligadas y determinar si la definición no posee errores de tipo. Si la sentencia es correcta, deberá presentar por pantalla los tipos de todas las entidades ligadas y si la sentencia presenta algún error, deberá indicarlo.

2. Consideraciones del proyecto

Este proyecto se hará también de manera grupal. La comisión de trabajo estará formada por los mismos integrantes del proyecto 1.

Los pasos o etapas a cumplir para el desarrollo del proyecto son:

1. Identificar los componentes léxicos del lenguaje y sus patrones asociados. Nótese que la gramática dada ya tiene los aspectos léxicos eliminados.

2. Identificar y resolver los problemas que presenta la gramática dada para la implementación de un analizador sintáctico LALR(1) para el lenguaje de la misma.

3. Dar una **definición dirigida por la sintaxis** que realice la determinación y el chequeo de tipos de una expresión para este lenguaje. Se pide una DDS que incluya **sólo** las acciones semánticas de la parte de expresiones.

4. Implementar el lenguaje utilizando JLex y CUP. Este paso implica la construcción de un analizador léxico en JLex y un analizador sintáctico en CUP. Al analizador sintáctico implementado, se le realizará una extensión semántica para que determine correctamente el tipo asociado a cada identificador que se liga, realice el chequeo de tipos y reporte los errores que se encuentren.

5. En la documentación del proyecto, además de los primeros tres puntos mencionados anteriormente, también debe incluirse una descripción general de la aplicación y una explicación de los métodos y estructuras auxiliares implementados y/o usados.

El resultado de la interpretación será una secuencia de ligaduras, o un mensaje de error. El mensaje de error (a excepción de los errores sintácticos) deberá especificar claramente el tipo de error ocurrido (Léxico o Semántico), el problema encontrado y el número de línea donde ocurrió. El código fuente se tomará del teclado (entrada estándar).