Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-422 — Compiladores

Apunte

Mapa del Proyecto

1. Pequeña Guía

En el repositorio `https://github.com/compiladores-lcc/compiladores2023` están los fuentes iniciales del proyecto. Al ejecutarlo (`cabal run`) podemos acceder al intérprete y evaluar expresiones dentro del mismo (similar a `ghci`).

```
FD4> let (x : Nat) = 20 in x + 10
30 : Nat
```

El proyecto se compone de los siguientes módulos:

- `Main.hs`: procesa la línea de comandos y en modo interactivo inicia la REPL.
- `Common.hs`: definiciones comunes, como `Pos`, que representa una posición en un archivo fuente.
- `Elab.hs`: elaboración desde sintaxis superficial a sintaxis "core".
- `Errors.hs`: tipo de mensajes de error.
- `Eval.hs`: evaluador big-step recursivo.
- `Global.hs`: Entorno global.
- `Lang.hs`: Definición de términos y declaraciones, tanto superficiales como "core".
- `MonadFD4.hs`: mónada de estado+error+IO usada en el compilador.
- `PPrint.hs`: pretty-printer. Pasa de sintaxis "core" a sintaxis superficial para poder imprimir.
- `Parse.hs`: Parser (y Lexer)
- `Subst.hs`: implementación de sustitución y funciones auxiliares de locally nameless.
- `TypeChecker.hs`: chequeador de tipos.

2. Ejercicios

Ej. 1.

a) ¿En qué módulo se encuentra definido el AST del lenguaje?

b) ¿Cuál es la diferencia entre un `Term` y un `TTerm`?

Ej. 2. ¿Cómo se agrega un comentario en código FD4? ¿Cómo puede cambiarse esto o agregar nuevos tipos de comentarios?

Ej. 3. La función `Parse.declOrTm` parsea, o bien una declaración top-level, o bien un término, y es usada desde el intérprete. ¿Por qué es necesario usar un `try` en su definición?

Ej. 4. La función `elab :: STerm -> Term` transforma términos superficiales a términos (core) y está definida en términos de una función auxiliar `elab'` que lleva un argumento extra `env` de tipo `[Name]`. ¿Cómo y para qué usa `elab'` ese argumento extra?

Ej. 5. La clase `MonadFD4` no tiene ninguna función miembro, pero importa las funciones miembro de sus superclases. ¿Cuáles funciones importa?

Ej. 6. En la función `Subst.open`, hay un `abort`. ¿Por qué es correcto esto? ¿Puede darse esa condición?

Ej. 7. ¿Qué funciones exporta el módulo `PPrint` y para qué sirven?

Ej. 8. Para poder hacer pretty-printing de un término se vuelve a una representación con nombres. Habíamos visto que al abrir un término con un nombre n es necesario que ese nombre no se encuentre libre en el término. ¿Cómo hace la función `openAll` para asegurarse que así sea?

3. Para charlar

Ej. 9. Para agregar alguna nueva construcción de términos al lenguaje, ¿qué componentes deben modificarse? Puede ser útil considerar algunos ejemplos distintos:

a) Negación lógica ($!0 = 1$, y $!n = 0$ para $n \neq 0$)

b) Un operador de multiplicación '$*$'

c) Condicional con guardas (á la Dijkstra, o á la Erlang). E.g.

```
if
  cond1 -> t1;
  cond2 -> t2;
  ...
  condn -> tn;
end
```

Ej. 10. Supongamos que queremos llevar alguna información de *profiling* del compilador. Por ejemplo, cuánto tiempo tardó en chequearse el tipo de cada parte de un término. ¿Cómo podemos hacer este cambio minimizando el impacto a código existente?

Ej. 11. Supongamos que queremos agregar un "operador" (por llamarlo de alguna manera) `__debug`$(-)$. El mismo puede usarse sobre cualquier expresión del programa, sin cambiar su semántica estática ni dinámica (i.e. ni el tipo ni su ejecución). Su función es, durante la compilación, imprimir información útil sobre el término en cuestión (e.g. su entorno léxico, tipo inferido, etc). ¿Cómo lo implementaría?