Capítulo 1

Sintaxis y semántica (boceto)

1.1. Sintaxis de CABS (Still in development)

Por el momento véase la especificación en el repositorio del compilador. Para sentar las bases de la notación que se usarán de ahora en adelante para definir la semántica diremos que **P** es un programa de nuestro lenguaje CABS formado por instrucciones globales como definición de variables globales (del estilo t var; con tipo y nombre de variable) y definición de funciones.

Las funciones estarán formadas por un tipo, un nombre de función, una lista de argumentos (posiblemente vacia), un cuerpo con instrucciónes $S \in \mathbf{Stm}$ y una expresión de retorno. Todo sigue una sintaxis sencilla C-like.

Las instrucciones S de un programa son las típicas de un lenguaje imperativo con asignaciones, operaciones aritmetico lógicas, instrucciones de control y, la joya de la corona, un **thread** para la ejecución de funciones con concurrencia imitando el comportamiento del fork en C.

1.2. Semántica de CABS (faltan arrays!!)

A continuación pasamos a hablar de la semántica del lenguaje CABS.

1.2.1. Preámbulo semántico

Antes de hablar de la semántica propiamente dicha necesitamos definir unas estructuras que nos serán posteriormente de gran utilidad.

En primer lugar definimos el conjunto de valores $\mathbb{V} = \mathbb{Z} \cup \mathbb{B}$ como la unión de los enteros y de los booleanos.

Una primera parte de la representación del estado estará formada por las variables globales de nuestro programa. Definimos $\mathbf{G} = \mathbf{Var} \hookrightarrow \mathbb{V}$ el conjunto de funciones parciales del conjunto de variables al conjunto de valores. El estado actual de nuestras variables

globales será una función de este conjunto y por lo general nos referiremos a ella con la letra G. Posteriormente cuando hablemos de variables locales también las definiremos como un conjunto de funciones similares pero que trataremos por separado por comodidad.

Por otro lado nos encontramos en nuestro lenguaje con la necesidad de definir un conjunto que recoja la información básica de las funciones y procedimientos. Definimos $\mathbf{F} = \mathbf{Func} \hookrightarrow (\mathbf{T} \times \mathbf{Stm} \times \mathbf{Args} \times (\mathbf{Exp} \cup \{\varepsilon\}))$ el conjunto de funciones parciales que asocia un nombre de función a su definición. Una función quedará definida por su tipo de retorno $t \in \mathbf{T}$, su código $S \in \mathbf{Stm}$, sus argumentos de entrada $\text{arg} \in \mathbf{Args}$ y su expresión de retorno $e \in \mathbf{Exp} \cup \{\varepsilon\}$. Según las necesidades, la expresión de retorno será una expresión booleana, una expresión aritmética o simplemente será la expresión vacia, empleada para los procedimientos.

Para dar un significado a nuestro programas tenemos que definir qué va a representar para nosotros su estado de ejecución. Definimos el conjunto de estados $\mathbf{State} = \mathbf{G} \times \mathbf{F} \times \mathbf{RP}$ como una tupla que recoja la información de las variables globales y de las funciones definidas en nuestro programa $\mathbf{P}$ así como una pila de marcos de ejecución o runtime processes que contendrá la información local de los procesos (e.g. variables locales (¡pilas de ámbitos locales!), código y tipo (¿y expresión de retorno?)).

La idea a seguir para definir nuestra semántica será apoyarnos en dos funciones auxiliares **init** y **start** que respectivamente inicializarán el estado global del programa y lanzarán a ejecución la función inicial *main*. Pasemos pues a definir la primera de ellas.

La función init.

Definimos la función $\mathbf{init} : \mathbf{Prog} \hookrightarrow \mathbf{State}$ de forma recursiva del siguiente modo.

$$
\begin{aligned}
\mathbf{init}(\varepsilon) &= (\mathbf{nil}, \mathbf{nil}, []) \\
\mathbf{init}(\mathit{int}\ \text{var};\ \mathbf{P}) &= (G\,[\ \text{var} \mapsto 0]\,, F, RP) \text{ donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{bool}\ \text{var};\ \mathbf{P}) &= (G\,[\ \text{var} \mapsto \textsc{false}]\,, F, RP) \text{ donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{int}\,\mathbf{func}(\text{arg})\{S;\ \mathbf{return}\ a\}\mathbf{P}) &= (G, F\,[\mathbf{func} \mapsto (\mathit{int}, S, \text{arg}, a)]\,, RP) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{bool}\,\mathbf{func}(\text{arg})\{S;\ \mathbf{return}\ b\}\mathbf{P}) &= (G, F\,[\mathbf{func} \mapsto (\mathit{bool}, S, \text{arg}, b)]\,, RP) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP}) \\
\mathbf{init}(\mathit{void}\,\mathbf{func}(\text{arg})\{S\}\mathbf{P}) &= (G, F\,[\mathbf{func} \mapsto (\mathit{void}, S, \text{arg}, \varepsilon)]\,, RP) \\
&\quad \text{donde } \mathbf{init}(\mathbf{P}) = (\mathbf{G}, \mathbf{F}, \mathbf{RP})
\end{aligned}
$$

La función (regla) start.

Definimos la función $\mathbf{start} : \mathbf{State} \hookrightarrow \mathbf{State}$ como

$$\mathbf{start}((\mathbf{G}, \mathbf{F}, \mathbf{RP})) = (\mathbf{G}, \mathbf{F}, [(\mathbf{nil} : [], S)]) \text{ donde } \mathbf{F}(\mathit{main}) = (\mathit{int}, S, \text{arg}, 0)$$

Con esto conseguimos crear un nuevo marco de ejecución con el código inicial de main. Nótese que la función inicializa la pila de ambitos de variables locales con el ambito **nil** que no contiene ninguna variable inicializada.

Semántica (Expresiones aritmético-lógicas).

(Primero hablemos de las funciones aritméticas que ya son jaleo. Se trata de expresiones con operandos, numerales y variables junto con funciones (por el momento como si no tuvieran argumentos!!!)).

Tenemos que definir una función (parcial) que dada una expresión (aritmética reducida) nos devuelva otra expresión más simplificada, pudiendo emplear un estado para ello (permitiendo modificaciones del mismo), hasta eventualmente quedarnos con un valor (entero por el momento).

Buscamos definir la función semántica $\mathcal{A} : (\mathbf{Aexp} \times \mathbf{State}) \hookrightarrow (\mathbf{Aexp} \times \mathbf{State})$