SimpleSem - SimplON

1 - Introducción

En este documento se detallan las características de SimpleSem, el procesador abstracto utilizado en la materia como medio para estudiar cómo implementar lenguajes de programación. En particular, SimpleSem cuenta con su implementación SimplON, que mediante una interfaz gráfica brinda la posibilidad de visualizar la memoria y registros del procesador abstracto en todo momento de la ejecución.

El documento se centra en la descripción de cómo son los programas e instrucciones que puede ejecutar y cuál es su efecto en el procesador abstracto. En primer lugar, se describe cómo es la estructura del procesador. Seguidamente se presenta cómo es la sintaxis de los programas. Luego se explica cuál es el efecto de ejecutar cada instrucción de SimpleSem. Posteriormente se mencionan características de funcionamiento del procesador y finalmente algunos detalles de su implementación e interfaz gráfica.

2 - Componentes y Funcionamiento del procesador SimpleSem

El procesador SimpleSem está compuesto por tres unidades de memoria:

- **C** la memoria de código, donde están almacenadas las instrucciones.
- **D** la memoria de datos, donde se realizan cálculos, se almacenan datos estáticos y registros de activación.
- **H** la memoria dinámica o *heap*, donde se almacena objetos o datos dinámicos.

Además, cuenta con cuatro registros especiales:

- ***Actual***: utilizado normalmente para indicar el comienzo del registro de activación actual en la memoria **D**.
- ***Libre:*** utilizado normalmente para indicar a partir de qué celda comienzan las celdas de memoria libre en **D**.
- ***po:*** utilizado normalmente para indicar la primera celda en **H** donde se puede construir un objeto o estructura.
- ***pc***: es utilizado para indicarle al procesador la instrucción actual a ejecutar.

El procesador abstracto de SimpleSem tiene un comportamiento muy sencillo. Básicamente toma la instrucción referenciada actualmente por el ***pc*** y si es halt[1] termina, sino la ejecuta. Si la instrucción ejecutada no produjo cambios en el ***pc,*** entonces lo incrementa en uno.

[1]Esta instrucción será explicada más adelante.

3 - Sintaxis

3.1 - Aspectos Léxicos

Los componentes léxicos se corresponden con las palabras válidas del lenguaje. En SimpleSem caracterizamos los siguientes tipos de palabras válidas: palabras clave, números, identificadores, operadores y anotaciones. A continuación, se presentan estos tipos y otras características léxicas de SimpleSem.

3.1.1 - Espacios en Blanco y Separadores

Los espacios en blanco en SimpleSem, simplemente hacen más fácil la lectura del programa por un humano y no son *tokens*. Un espacio en blanco es un espacio, tab o un salto de línea.

3.1.2 - Comentarios

Los comentarios son secuencias de caracteres que comienzan con un % y terminan con un salto de línea. Son como los comentarios de línea simple de los lenguajes de programación vistos en la carrera.

3.1.3 - Palabras Clave

Las palabras clave de SimpleSem se corresponden con los nombres de las instrucciones, los registros rápidos y las memorias. Las correspondientes a las instrucciones son: **SetD**, **SetH**, **SetActual**, **SetLibre**, **SetIn**, **SetOut**, **SetLabel**, **SetPO**, **JumpT**, **Jump** y **Halt**. Las vinculadas a los registros rápidos son: **Actual**, **Libre**, **pc**. Las vinculadas a las memorias son: **D** y **H**.

3.1.4 - Números

Un número en SimpleSem es una secuencia de dígitos. A estas secuencias las notaremos como ***números***. Los números son utilizados para dar valor y direccionar las celdas de memoria.

3.1.5 - Identificadores

Un identificador en SimpleSem es una secuencia de caracteres que comienza con una letra y continúa con letras o dígitos. A estas secuencias las notaremos como ***identificadores***. Los identificadores son utilizados para nombrar etiquetas en el lenguaje.

3.1.6 - Operadores y Separadores

En SimpleSem se cuenta con siguientes operadores: **+**, **-**, *****, **/**, **==**, **!=**, **<**, **>**, **<=**, **>=**, **!**, **|** y **&**. El significado de estos operadores es el usual de los lenguajes de programación modernos. Los separadores son **[**, **]**, **(** y **)**. Los corchetes son utilizados para indizar los accesos a memoria y los paréntesis son utilizados como separadores matemáticos.

3.1.7 - Anotaciones

Una anotación en SimpleSem es una secuencia de caracteres que comienza con **@** y continúa con una secuencia de caracteres hasta llegar a un salto de línea o un comentario. A estas secuencias las

notaremos como ***anotaciones***. Las anotaciones son utilizadas por el entorno gráfico para mostrar mensajes asociados a las locaciones de memoria actualizadas.

3.1.8 - Insensiblidad a las Mayúsculas

Los componentes léxicos de SimpleSem no son sensibles a las mayúsculas. Por lo tanto, por ejemplo, **SetD** es equivalente a **setd**, o equivalente a **sETd**.

3.2 - Gramática

A continuación, se presenta la gramática de SimpleSem, la cual caracteriza a los programas sintácticamente válidos del lenguaje. Note que se utiliza la notación de BNF-Extendida donde el superíndice **+** indica repetición de una o más veces del símbolo, y el superíndice **?** indica que el símbolo es opcional (es decir, puede estar o no).

<Programa>	::=	<Instrucción>$^{+}$
<Instrucción>	::=	***Identificador***$^{?}$ <Signatura> **anotación**$^{?}$
<Signatura>	::=	**SetD** <Expresión>, <Expresión> | **SetH** <Expresión>, <Expresión> | **SetActual** <Expresión> | **SetLibre** <Expresión> | **SetIn** <Expresión> | **SetOut** <Expresión> | **SetPO** <Expresión> | **SetLabel** ***Identificador***, <Expresión> | **JumpT** <Expresión>, <Expresión> | **Jump** <Expresión> | **Halt**
<Expresión>	::=	<Expresión> <OpBinaria> <Expresión> | <OpUnaria> <Expresión> | <Operando>
<OpBinaria>	::=	**+** | **-** | ***** | **/** | **==** | **!=** | **<** | **>** | **<=** | **>=** | **&** | **|** |
<OpUnaria>	::=	**+** | **-** | **!**
<Operando>	::=	**D[**<Expresión>**]** | **H[**<Expresión>**]** | **Actual** | **Libre** | ***Numero*** | ***Identificador*** | **pc** | **(**<Expresión>**)**

4 - Semántica

4.1 - Etiquetas

Las etiquetas anteceden instrucciones y son utilizadas en general por las instrucciones de salto para mover el ***pc*** al lugar adecuado a ejecutar.

Cuando el programa a ejecutar en SimpleSem es ensamblado en la memoria **C**, se recopila toda la información referente a las etiquetas. A una etiqueta que antecede a una instrucción **I** se le asocia el valor de la locación de la memoria **C** en la que **I** fue almacenada.

Además, como se verá en la sección 4.3.8, en ejecución es posible asociar valores a nuevas etiquetas.

4.2 - Anotaciones

Las anotaciones no tienen ningún efecto sobre la ejecución del programa. Son utilizadas por la interfaz gráfica de simplON para añadir información visible en las celdas de la memoria **D** o **H**.

4.3 - Comportamiento de las instrucciones

A continuación se detalla el comportamiento de cada instrucción de SimpleSem.

4.3.1 - SetD Destino, Fuente

Es la instrucción utilizada para cambiar los valores almacenados en la memoria **D**. Esta instrucción evalúa la expresión ***Fuente*** y almacena su resultado en la memoria **D**, específicamente, en la dirección resultante de evaluar la expresión ***Destino***.

4.3.2 - SetH Destino, Fuente

Es la instrucción utilizada para cambiar los valores almacenados en la memoria **H**. Esta instrucción evalúa la expresión ***Fuente*** y almacena su resultado en la memoria **H**, específicamente, en la dirección resultante de evaluar la expresión ***Destino***.

4.3.3 - SetActual *Fuente*

Es la instrucción utilizada para cambiar el valor de **Actual**. Esta instrucción evalúa la expresión ***Fuente*** y asigna su resultado al registro **Actual**.

4.3.4 - SetLibre *Fuente*

Es la instrucción utilizada para cambiar el valor de **Libre**. Esta instrucción evalúa la expresión ***Fuente*** y asigna su resultado al registro **Libre**.

4.3.5 - SetIn Destino

Es la instrucción utilizada para leer valores ingresados por el usuario, que funciona como un *read* de enteros. La instrucción lee un valor de la entrada y lo almacena en la memoria **D**, específicamente, en la dirección resultante de evaluar la expresión ***Destino.***

4.3.6 - SetOut *Fuente*

Es la instrucción utilizada para mostrar valores por pantalla, que funciona como un *print* de enteros. La instrucción muestra por pantalla el resultado de evaluar la expresión ***Fuente***.

4.3.7 - SetPO *Fuente*

Es la instrucción utilizada para cambiar el valor de **po**. Esta instrucción evalúa la expresión ***Fuente*** y asigna su resultado al registro **po**.

4.3.8 - SetLabel Identificador, Fuente

Es la instrucción utilizada para setear el valor de un Label, la cual es útil para asociar un Label a una dirección de memoria (como por ejemplo, el comienzo de una VT). La instrucción crea una etiqueta (o cambia el valor de una existente) con nombre caracterizado por el ***Identificador*** y le asigna el valor resultante de evaluar la expresión ***Fuente***.

4.3.9 - JumpT Fuente, Condición

Es la instrucción utilizada para realizar saltos condicionales, modificando el ***pc***. La instrucción asigna el valor resultante de evaluar la expresión ***Fuente*** al registro ***pc*** siempre y cuando el valor de evaluar la expresión ***Condición*** sea verdadero.

4.3.10 - Jump Fuente

Es la instrucción utilizada para realizar saltos incondicionales, modificando el ***pc***. La instrucción asigna el valor resultante de evaluar la expresión ***Fuente*** al registro ***pc***.

4.3.11 - Halt

Es la instrucción utilizada para finalizar la ejecución del programa actualmente cargado.

4.4 - Evaluación de Expresiones

4.4.1 - Evaluando Expresiones Binarias y Unarias

Las expresiones son evaluadas de izquierda a derecha. Todos los operadores tienen la misma precedencia y, en general, su comportamiento es el usual en los lenguajes de programación modernos. En particular, los operadores que arrojan resultados booleanos retornan 1 en caso dar resultado verdadero y 0 en caso de dar falso. Además, los operadores que usualmente reciben valores booleanos (como **==** y **!=**) asumen que 0 es falso y cualquier valor distinto de 0 es verdadero.

4.4.2 - Evaluando Operandos

A continuación se indica el resultado de evaluar cada uno de los operandos detallados en la gramática:

D[<Expresión>]	Retorna el valor de la memoria **D** almacenado en la locación resultante de evaluar **<Expresión>**
H[<Expresión>]	Retorna el valor de la memoria **H** almacenado en la locación resultante de evaluar **<Expresión>**

Actual	Retorna el valor del registro **Actual**
Libre	Retorna el valor del registro **Libre**
pc	Retorna el valor del registro **pc**
po	Retorna el valor del registro **po**
Numero	Retorna el valor numérico de la secuencia de dígitos que compone al número
Identificador	Retorna el valor asociado a la etiqueta con el nombre caracterizado por la secuencia de caracteres que compone al identificador
(<Expresión>)	Retorna el valor resultante de evaluar **<Expresión>**

5 - SimplON

SimplON es la implementación de SimpleSem que utilizamos en la materia. Esta implementación cuenta con una interfaz gráfica que muestra el estado de las tres memorias y los valores de los registros. En particular, los registros son mostrados en la memoria en la que son utilizados. Además, SimplON se vale de las anotaciones que preceden a las instrucciones para decorar visualmente las celdas de las memorias. SimplON también muestra los valores asociados a las etiquetas que estaban en el programa, y a las generadas dinámicamente.

5.1 - Ejecutando SimplON

SimplON está implementado en Java y requiere de la versión 1.8 de la máquina virtual. Para correrlo sólo es necesario ejecutar el archivo "*SimplON.jar*".

Es posible configurar los tamaños máximos de las memorias **D** y **H** (por defecto, 1000 celdas cada una). Para esto se debe invocar a SimplON con dos números enteros como argumentos de entrada: el primer número corresponde al tamaño de la memoria **D**, mientras que el segundo al tamaño de la memoria **H**.

5.2 - Modos de Ejecución de un Programa en SimplON

SimplON brinda dos formas de ejecutar el programa cargado: "*Completa"* o "*Paso a Paso*". La ejecución completa aplica el algoritmo de ejecución de SimpleSem según fue comentado en la Sección 2, interrumpiéndose en caso de requerir Input de usuario. Por otra parte, la ejecución paso a paso ejecuta un ciclo del algoritmo cada vez que se presiona el botón "*Step*". Este último modo es el que permite ver la evolución de las memorias, registros, etiquetas y anotaciones a medida que se ejecuta el programa cargado.

5.3 - Anotaciones en SimplON

Las anotaciones son una herramienta para decorar las celdas de memoria, de manera que sea fácil visualizar qué elementos de la traducción/implementación están vinculados los valores asociados a la celda decorada. Por lo tanto, es común crear anotaciones para enlaces dinámicos, punteros de retorno, variables, parámetros, atributos, etc.

A efectos prácticos, cuando se ejecuta una instrucción con una anotación, SimplON analiza cuál fue la celda de memoria afectada por esa ejecución y agrega la anotación a esa celda de memoria. Una celda de memoria puede estar decorada por una única anotación a la vez. Además, siguiendo la intuición detrás del uso del registro **Libre**, cada vez que dicho registro reduce su valor (es decir, cuando se está "limpiado" memoria) se eliminan todas las anotaciones de las celdas de memoria comprendidas entre el valor viejo y valor nuevo del registro.