Semántica Dinámica

Semántica Operacional

- Semántica Operacional
  - Describir el significado de una sentencia o programa por la especificación de los efectos de ejecutarlo en una “máquina“. El cambio de estado de máquina define el significado de sentencia.
  - Considere el estado de la máquina como el conjunto de valores en la jerarquía de almacenamiento de la máquina (memoria, registros, etc).
  - Semántica operacional natural: resultado final de la ejecución es lo que importa.
  - Semántica operacional estructural: cambios de estado a cada paso.
- Para usar semántica operacional en un lenguaje de alto nivel, es necesaria una máquina virtual.

Semántica Operacional

- Un interpretador puro de hardware sería muy caro
- Un interpretador puro de software también tiene problemas
  - Las caracteristicas detalladas de un computador dejaria acciones dificiles de entender
  - Una definición semántica seria dependiente de la máquina
- Una mejor alternativa: una completa simulación
- El proceso:
  - Construir un interpretador (transforma código de origen en código de máquina de una computador idealizado)
  - Construir un simulador para el computador idealizado

Semántica Operacional

C Statement

```
for (expr1; expr2; expr3) {
   ...
}
```

Meaning

```
        expr1;
loop:   if expr2 == 0 goto out
        ...
        expr3;
        goto loop
out:    ...
```

Semántica Operacional

- Usos de semánticas operacionales:
  - Manuales de lenguaje y libros de texto de programación
  - Enseñanza de lenguajes de programación
- Dos niveles diferentes de uso deuna semántica operacional:
  - Semántica operacional natural
  - Semántica operacional estructural
- Evaluación
  - Buena si usada informalmente
  - Extremadamente compleja si usada formalmente

Semánticas denotacionales

- Basada en la teoría de funciones recursivas
- El método de descripción semántica más abstracto en relación a semántica operacional
- Originalmente desarrollada por Scott y Strachey (1970)

Semánticas denotacionales

- El proceso de construir una especificación denotacional para un lenguaje:
  - Define un objeto matemático para cada entidad del lenguaje
    - Imagine entidad como cualquier no terminal o terminal de la descripción del lenguaje
    - Por ejemplo, <id>, <expr>, etc.
  - Define una función que mapea las instancias de esa entidad de lenguaje para instancias del objeto matemático correspondiente
    - Dominio sintáctico: cadenas de símbolos gramaticales de las entidades.
    - Dominio semántico: objetos matemáticos.
- El significado de la construcción de lenguaje es definido apenas por los valores de las variables de programas
  - No tendremos el paso a paso del programa (semántica operacional)
  - Apenas las alteraciones en sus variables

Semánticas denotacionales: estado de un programa

- El estado de una programa son los valores actuales de todas sus variables

$$s = \{\langle i_1, v_1\rangle, \langle i_2, v_2\rangle, \ldots, \langle i_n, v_n\rangle\}$$

- Considere VARMAP una función que, cuando dado un nombre de variable y estado, retorna el actual valor de la variable

$$\text{VARMAP}(i_j, s) = v_j$$

- Podemos imaginar la ejecución de un programa como sucesivas instancias del conjunto s
- La dificultad, en un programa moderno, es tener percepción de cuantas variables realmente existen.

Dos ejemplos simples

- Considere la gramática:
- `<bin_num> → '0'`
- `| '1'`
- `| <bin_num> '0'`
- `| <bin_num> '1'`
- Hacemos el cálculo del valor por la gramática de atributos, vamos a usar ahora la semántica denotacional
- Para ‘110‘, cual sería el arbol?
- Para la cadena ‘1‘, RHS de la primera regla, ¿cuál símbolo matemático debo asociar?

Dos ejemplos simples

- La función es definida de la siguiente forma:

M_{bin}('0') = 0
M_{bin}('1') = 1
M_{bin}(<bin_num> '0') = 2 * M_{bin}(<bin_num>)
M_{bin}(<bin_num> '1') = 2 * M_{bin}(<bin_num>) + 1

- También podemos anotar el árbol con los valores de los objetos matemáticos

Números decimales

```
<dec_num> → '0' | '1' | '2' | '3' | '4' | '5' |
            '6' | '7' | '8' | '9' |
            <dec_num> ('0' | '1' | '2' | '3' | '4' |
                       '5' | '6' | '7' | '8' | '9')
```

M_{dec}('0') = 0, M_{dec}('1') = 1, M_{dec}('2') = 2, ..., M_{dec}('9') = 9

M_{dec}(<dec_num> '0') = 10 * M_{dec}(<dec_num>)

M_{dec}(<dec_num> '1') = 10 * M_{dec}(<dec_num>) + 1

...

M_{dec}(<dec_num> '9') = 10 * M_{dec}(<dec_num>) + 9

Expresiones

- Vamos a definir la función M_e, que recibe una expresión y retorna el objeto matemático.
- Hasta ahora, los ejemplos fueron simples, sin ninguna entidad de lenguaje con más alto nivel
- Expresiones forman el conjunto $\mathbb{Z} \cup \{error\}$
- Asumimos que expresiones son número decimales, variables o expresiones binarias teniendo un operador aritmético y dos operandos, cada uno puede ser una expresión:

```
<expr>          → <dec_num> | <var> | <binary_expr>
<binary_expr>   → <left_expr> <operator> <right_expr>
<left_expr>     → <dec_num> | <var>
<right_expr>    → <dec_num> | <var>
<operator>      →  + | *
```

Expresiones

M_e(<expr>, s) Δ= case <expr> of
<dec_num> => M_{dec}(<dec_num>, s)
<var> => if VARMAP(<var>, s) == **undef**
then **error**
else VARMAP(<var>, s)
<binary_expr> =>
if (M_e(<binary_expr>.<left_expr>, s) == **undef** OR
M_e(<binary_expr>.<right_expr>, s) == **undef**)
then **error**
else if (<binary_expr>.<operator> == '+')
then M_e(<binary_expr>.<left_expr>, s) +
M_e(<binary_expr>.<right_expr>, s)
else M_e(<binary_expr>.<left_expr>, s) *
M_e(<binary_expr>.<right_expr>, s)

Sentencias de asignación

- Funciones de mapeamiento mapean listas de sentencias y estados para estados U {*error*}

$$
\begin{aligned}
M_a(x = E, s)\ \Delta = &\ \text{if } M_e(E, s) == \textbf{error} \\
&\ \text{then } \textbf{error} \\
&\ \text{else } s' = \{\langle i_1, v_1' \rangle, \langle i_2, v_2' \rangle, \ldots, \langle i_n, v_n' \rangle\}, \text{ where} \\
&\quad \text{for } j = 1, 2, \ldots, n \\
&\quad\quad \text{if } i_j == x \\
&\quad\quad\quad \text{then } v_j' = M_e(E, s) \\
&\quad\quad\quad \text{else } v_j' = \text{VARMAP}(i_j, s)
\end{aligned}
$$

Lazos lógicos con pre-testeo

- Funciones de mapeamiento mapean listas de sentencias y estados para estados U {*error*}
- Vamos a suponer que existen M_{sl} (listas de sentencias) y M_b (expresiones booleana)

$$
\begin{aligned}
M_l(\texttt{while}\ B\ \texttt{do}\ L, s) \Delta= \text{if } M_b(B, s) == \textbf{undef} \\
&\text{then } \textbf{error} \\
&\text{else if } M_b(B, s) == \text{false} \\
&\quad \text{then } s \\
&\quad \text{else if } M_{sl}(L, s) == \textbf{error} \\
&\quad\quad \text{then } \textbf{error} \\
&\quad\quad \text{else } M_l(\texttt{while}\ B\ \texttt{do}\ L, M_{sl}(L, s))
\end{aligned}
$$

Significado del lazo

- El significado del lazo es el valor de las variables del programa después de que las sentencias en el lazo hayan sido ejecutadas el número de veces necesario, asumiendo que ocurrieran errores
- Esencialmente, el lazo fue convertido de iteración para recursión, donde el control de la recursión es definido matemáticamente por otras funciones recursivas de mapeamiento de estado
  - La recursión es más fácil de describir con rigor matemático que la iteración
  - Pero también exige un esfuerzo mental mayor para comprensión

Evaluación de la semántica denotacional

- Puede ser usada para probar la correción de programas
- Sugiere una manera rigurosa para pensar en los programas
- Puede ser una ayuda al diseño del lenguaje
- Ha sido usada en sistemas de generación de compilador
- Debido a su complejidad, es de poco uso para usuarios de lenguaje