Análisis de Lenguajes de Programación

Lautaro Capezio, Luciano Duarte e Ignacio Basualdo

Septiembre 2025

Ejercicio 1: Extensión de Sintaxis

Para extender el lenguaje LIS, se modifican las sintaxis de expresiones enteras (`intexp`) para añadir el operador de incremento, y la sintaxis de comandos (`comm`) para añadir el comando de selección múltiple `case`.

Sintaxis Abstracta

Listing 1: Gramática Abstracta Extendida

```
-- Expresiones Enteras
intexp ::= nat | var | -u intexp
     | intexp + intexp
     | var ++
     | intexp -b intexp
     | intexp * intexp
     | intexp / intexp
```

Sintaxis Concreta

La sintaxis concreta define la representación textual. Aquí sí se añade una regla para poder escribir el comando `case`.

Listing 2: Gramática Concreta Extendida

```
intexp ::= nat
     | var
     | var '++'
     | '-' intexp
     | intexp '+' intexp | intexp '-' intexp
     | intexp '*' intexp | intexp '/' intexp
     | '(' intexp ')'

comm ::= skip
     | var '=' intexp
     | comm ';' comm
     | 'if' boolexp '{' comm '}'
     | 'if' boolexp '{' comm '}' 'else' '{' comm '}'
     | 'repeat' '{' comm '}' 'until' boolexp
     | 'case' '{' boolexp ':' '{' comm '}' commCase '}'

commCase ::= skip
     | boolexp ':' '{' comm '}' commCase
```

Ejercicio 4: Semántica Big-Step para ++

El operador `v++` tiene un doble efecto: su valor resultante es el valor de `v` más uno y, como efecto secundario, el estado se actualiza con este nuevo valor.

Para modelar esto, se añade la siguiente regla de inferencia a la semántica operacional big-step.

$$\frac{x \in \text{dom } \sigma}{\langle x++, \sigma \rangle \Downarrow_{\text{exp}} \langle \sigma x + 1, [\sigma | x : \sigma x + 1] \rangle} \ VarInc$$

Ejercicio 5: Determinismo de la Relación de Transición

La pregunta a responder es si la relación de evaluación de un paso ($\leadsto$) es determinista. La respuesta es **sí**.

Propiedad a Probar: Se debe demostrar que para cualquier configuración t, si existen dos transiciones posibles t' y t'' a partir de t, entonces estas deben ser idénticas. Formalmente:

$$\forall t, t', t''.(t \leadsto t' \land t \leadsto t'') \Rightarrow t' = t''$$

La prueba se realiza por inducción estructural sobre la forma del comando c en la configuración $t = \langle c, \sigma \rangle$. Se asume que la relación de evaluación de expresiones $\Downarrow_{\text{exp}}$ es determinista.

Caso Base (Comando `v = e`, Regla ASS) Sea $t = \langle v = e, \sigma \rangle$. La única regla aplicable es ASS. Si $t \leadsto t'$ y $t \leadsto t''$, entonces:

- $\langle e, \sigma \rangle \Downarrow_{\text{exp}} \langle n, \sigma' \rangle$ tal que $t' = \langle \text{skip}, \sigma'[v : n] \rangle$.
- $\langle e, \sigma \rangle \Downarrow_{\text{exp}} \langle n_2, \sigma_2 \rangle$ tal que $t'' = \langle \text{skip}, \sigma_2[v : n_2] \rangle$.

Dado que $\Downarrow_{\text{exp}}$ es determinista, se tiene que $n = n_2$ y $\sigma' = \sigma_2$. Por lo tanto, $t' = t''$.

Caso Base (Comando `skip;` c_1, Regla SEQ$_1$) Sea $t = \langle \text{skip}; c_1, \sigma \rangle$. La única regla que puede aplicarse a esta forma es SEQ$_1$. La regla SEQ$_2$ no puede aplicarse, ya que requeriría una transición para $\langle \text{skip}, \sigma \rangle$, la cual no existe. Por lo tanto, cualquier transición desde t debe ser $\langle c_1, \sigma \rangle$. Se concluye que $t' = t''$.

Caso Base (Comando `if b ...`, Reglas IF$_1$/IF$_2$) Sea $t = \langle \texttt{if } b \texttt{ then } c_0 \texttt{ else } c_1, \sigma \rangle$. La transición depende de la evaluación de b. Si se usa la regla IF$_1$ para obtener t', es porque $\langle b, \sigma \rangle \Downarrow_{\text{exp}} \langle \textbf{true}, \sigma' \rangle$, y $t' = \langle c_0, \sigma' \rangle$. Para obtener una transición distinta t'', se debería poder aplicar la regla IF$_2$, lo que requeriría que $\langle b, \sigma \rangle \Downarrow_{\text{exp}} \langle \textbf{false}, \sigma' \rangle$. Esto contradice la deterministicidad de $\Downarrow_{\text{exp}}$. Por lo tanto, solo una de las reglas es aplicable, y la transición es única.

Caso Base (Comando `repeat ...`, Regla REPEAT) Sea $t = \langle \texttt{repeat } c \texttt{ until } b, \sigma \rangle$. La única regla aplicable es REPEAT, que transforma t de forma única en $\langle c; \texttt{if } b \texttt{ then skip else repeat } c \texttt{ until } b, \sigma \rangle$. Por lo tanto, $t' = t''$.

Caso Inductivo (Comando c_0; c_1) Sea $t = \langle c_0; c_1, \sigma \rangle$, con $c_0 \neq \text{skip}$. Asumimos que existen dos transiciones $t \leadsto t'$ y $t \leadsto t''$.

Dado que $c_0 \neq \text{skip}$, la única regla que puede haberse aplicado para la primera transición $t \leadsto t'$ es SEQ$_2$. Por lo tanto, sabemos que:

- Existe una transición $\langle c_0, \sigma \rangle \leadsto \langle c_0', \sigma' \rangle$.
- Y t' tiene la forma $t' = \langle c_0'; c_1, \sigma' \rangle$.

Ahora, analizamos los dos casos posibles para la segunda transición $t \leadsto t''$:

Posibilidad 1: Se utilizó la regla SEQ$_1$ Si se hubiera utilizado la regla SEQ$_1$, esto implicaría que el comando original $c_0; c_1$ tendría la forma $\text{skip}; c_1$, lo que significa que $c_0 = \text{skip}$. Esto contradice nuestra suposición inicial. **Absurdo!**

Posibilidad 2: Se utilizó la regla SEQ$_2$ Si se utiliza la regla SEQ$_2$, entonces debe existir otra transición para el subcomando:

- $\langle c_0, \sigma \rangle \leadsto \langle c_0'', \sigma'' \rangle$.
- Y t'' tiene la forma $t'' = \langle c_0''; c_1, \sigma'' \rangle$.

Ahora tenemos dos transiciones que parten de la misma configuración $\langle c_0, \sigma \rangle$. Como c_0 es un subcomponente estructural, podemos aplicar la **Hipótesis Inductiva (HI)**.

Por HI, la transición de $\langle c_0, \sigma \rangle$ es determinista, lo que nos obliga a concluir que:

$$\langle c_0', \sigma' \rangle = \langle c_0'', \sigma'' \rangle$$

Dado que las partes componentes de t' y t'' son idénticas, se sigue directamente que $\mathbf{t}' = \mathbf{t}''$.

Ejercicio 6: Prueba de Equivalencia Semántica

a) Demostración para `x = x + 1; y = x`

Para probar la ejecución, se construye un árbol de derivación para cada paso de la secuencia de comandos.

Paso 1: Ejecución de `x = x + 1` Primero, se deriva la ejecución del primer comando. Se define un estado intermedio $\sigma_1 = [\sigma | x : \sigma x + 1]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{x \in \text{dom } \sigma}{\langle x, \sigma \rangle \Downarrow_{\text{exp}} \langle \sigma x, \sigma \rangle} \text{VAR} \quad \dfrac{}{\langle 1, \sigma \rangle \Downarrow_{\text{exp}} \langle 1, \sigma \rangle} \text{NVAL}}{\langle x + 1, \sigma \rangle \Downarrow_{\text{exp}} \langle \sigma x + 1, \sigma \rangle} \text{PLUS}}{\langle x = x + 1, \sigma \rangle \rightsquigarrow \langle \text{skip}, \sigma_1 \rangle} \text{ASS}}{\langle x = x + 1; y = x, \sigma \rangle \rightsquigarrow \langle \text{skip}; y = x, \sigma_1 \rangle} \text{SEQ}_2$$

Paso 2: Ejecución de `skip; y = x` Luego, se consume el `skip` con la regla SEQ_1.

$$\dfrac{}{\langle \text{skip}; y = x, \sigma_1 \rangle \rightsquigarrow \langle y = x, \sigma_1 \rangle} \text{SEQ}_1$$

Paso 3: Ejecución de `y = x` Finalmente, se ejecuta el segundo comando en el estado intermedio σ_1. Se define el estado final $\sigma' = [\sigma_1 | y : \sigma_1 x]$.

$$\dfrac{\dfrac{x \in \text{dom } \sigma_1}{\langle x, \sigma_1 \rangle \Downarrow_{\text{exp}} \langle \sigma_1 x, \sigma_1 \rangle} \text{VAR}}{\langle y = x, \sigma_1 \rangle \rightsquigarrow \langle \text{skip}, \sigma' \rangle} \text{ASS}$$

Resumen de la Ejecución: La secuencia completa de transiciones es la siguiente.

$$\begin{aligned} \langle x = x + 1; y = x, \sigma \rangle &\rightsquigarrow \langle \text{skip}; y = x, [\sigma | x : \sigma x + 1] \rangle \\ &\rightsquigarrow \langle y = x, [\sigma | x : \sigma x + 1] \rangle \\ &\rightsquigarrow \langle \text{skip}, [\sigma | x : \sigma x + 1, y : \sigma x + 1] \rangle \end{aligned}$$

Por lo tanto, se demuestra la ejecución completa usando la clausura transitiva de la relación:

$$\langle x = x + 1; y = x, \sigma \rangle \rightsquigarrow^* \langle \text{skip}, [\sigma | x : \sigma x + 1, y : \sigma x + 1] \rangle$$

b) Demostración para `y = x++` y Conclusión

Se deriva la ejecución del segundo programa. Primero, se utiliza la regla de evaluación para la expresión `x++`.

$$\dfrac{x \in \text{dom } \sigma}{\langle x{+}{+}, \sigma \rangle \Downarrow_{\text{exp}} \langle \sigma x + 1, [\sigma | x : \sigma x + 1] \rangle} \text{VARINC}$$

Por lo tanto, la derivación para el comando de asignación completo es:

$$\dfrac{\dfrac{x \in \text{dom } \sigma}{\langle x{+}{+}, \sigma \rangle \Downarrow_{\text{exp}} \langle \sigma x + 1, [\sigma | x : \sigma x + 1] \rangle} \text{VARINC}}{\langle y = x{+}{+}, \sigma \rangle \rightsquigarrow \langle \text{skip}, [\sigma | y : \sigma x + 1, x : \sigma x + 1] \rangle} \text{ASS}$$

Conclusión de Equivalencia Como se demostró en los apartados a) y b), ambos programas, partiendo de un mismo estado inicial σ, terminan en el mismo estado final.

- $\langle x = x + 1; y = x, \sigma\rangle \rightsquigarrow^* \langle \text{skip}, [\sigma | y : \sigma x + 1, x : \sigma x + 1]\rangle$
- $\langle y = x + +, \sigma\rangle \rightsquigarrow^* \langle \text{skip}, [\sigma | y : \sigma x + 1, x : \sigma x + 1]\rangle$

Se tiene entonces que ambos programas son **semánticamente equivalentes**.