Ayudantía 9

Semántica

Javier Larré

Pontificia Universidad Católica de Chile
javier.larre@uc.cl

08 de junio de 2026

Ayudantía de hoy

1. ¿Qué es semántica?
2. Tipos de semánticas
3. Ejercicios
4. Referencing

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.
- ¿Cuál es la sintaxis y la semántica de un loop en C? ¿Cuál es la diferencia con Python?

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.
- ¿Cuál es la sintaxis y la semántica de un loop en C? ¿Cuál es la diferencia con Python?
- En la tarea 1 de este curso tuvieron que desarrollar un parser a partir de una sintaxis.

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.
- ¿Cuál es la sintaxis y la semántica de un loop en C? ¿Cuál es la diferencia con Python?
- En la tarea 1 de este curso tuvieron que desarrollar un parser a partir de una sintaxis.
- En la tarea 2 tuvieron que definir la semántica de un lenguaje OOP.

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.
- ¿Cuál es la sintaxis y la semántica de un loop en C? ¿Cuál es la diferencia con Python?
- En la tarea 1 de este curso tuvieron que desarrollar un parser a partir de una sintaxis.
- En la tarea 2 tuvieron que definir la semántica de un lenguaje OOP.
- En la tarea 3...

¿Qué es la semántica?

- La **sintaxis** de un programa es la forma de sus expresiones.
- Su **semántica** es el significado de esas expresiones.
- ¿Cuál es la sintaxis y la semántica de un loop en C? ¿Cuál es la diferencia con Python?
- En la tarea 1 de este curso tuvieron que desarrollar un parser a partir de una sintaxis.
- En la tarea 2 tuvieron que definir la semántica de un lenguaje OOP.
- En la tarea 3...



¿Qué es la semántica?

Semántica operacional
The idea behind operational semantics is to describe the meaning of a statement or program by specifying the effects of running it on a machine. The effects on the machine are viewed as the sequence of changes in its state, where the machine's state is the collection of the values in its storage. An obvious operational semantics description, then, is given by executing a compiled version of the program on a computer.

¿Qué es la semántica?

Semántica operacional
The idea behind operational semantics is to describe the meaning of a statement or program by specifying the effects of running it on a machine. The effects on the machine are viewed as the sequence of changes in its state, where the machine's state is the collection of the values in its storage. An obvious operational semantics description, then, is given by executing a compiled version of the program on a computer.
[Sebesta, 2023]

Semántica axiomática
Axiomatic semantics, thus named because it is based on mathematical logic, is the most abstract approach to semantics specification discussed in this chapter. Rather than directly specifying the meaning of a program, axiomatic semantics specifies what can be proven about the program. Recall that one of the possible uses of semantic specifications is to prove the correctness of programs.

Preguntas

- ¿Por qué assembly no se puede usar para definir un *statement* en semánticas operacionales?
- ¿Qué es una regla de inferencia? ¿Cómo se define?
- Expresa `a = b / 2 - 1 {a < 10}` en la forma {P}S{Q}.

- Escribe, en instrucciones de máquina virtual, la definición semántica operacional de las siguientes estructuras:
  - for de C
  - switch de C
  - if-else de C++
  - do-while de Java

¿Por qué assembly no se puede usar para definir un *statement* en semánticas operacionales?

- La semántica operacional busca explicar el significado de un programa, expresiones o *statements* especificando los efectos sobre la máquina.

¿Por qué assembly no se puede usar para definir un *statement* en semánticas operacionales?

- La semántica operacional busca explicar el significado de un programa, expresiones o *statements* especificando los efectos sobre la máquina.
- Necesitamos que la descripción sea concisa y *human-readable* para una gran cantidad de operaciones que deben ser pequeñas, para luego construir programas complejos con una gran variedad de resultados. Además que estos programas pueden incluir operaciones I/O o de memoria.

¿Por qué assembly no se puede usar para definir un *statement* en semánticas operacionales?

- La semántica operacional busca explicar el significado de un programa, expresiones o *statements* especificando los efectos sobre la máquina.
- Necesitamos que la descripción sea concisa y *human-readable* para una gran cantidad de operaciones que deben ser pequeñas, para luego construir programas complejos con una gran variedad de resultados. Además que estos programas pueden incluir operaciones I/O o de memoria.
- Por lo tanto, se prefiere diseñar lenguajes de nivel intermedio e interpretadores para ”computadores ideales” en vez de lenguajes de máquina que dependan de alguna arquitectura.

Reglas de inferencia

- Es un método para inferir alguna verdad sobre algún predicado a partir de otros predicados.

Reglas de inferencia

- Es un método para inferir alguna verdad sobre algún predicado a partir de otros predicados.
- Se define de la siguiente manera: $\frac{S1, S2, \ldots, Sn}{S}$.

Reglas de inferencia

- Es un método para inferir alguna verdad sobre algún predicado a partir de otros predicados.
- Se define de la siguiente manera: $\frac{S1, S2, \ldots, Sn}{S}$.
- Donde $S1, S2, \ldots, Sn$ son predicados. A este conjunto se le puede decir hipótesis o antecedente.

Reglas de inferencia

- Es un método para inferir alguna verdad sobre algún predicado a partir de otros predicados.
- Se define de la siguiente manera: $\frac{S1, S2, \ldots, Sn}{S}$.
- Donde $S1, S2, \ldots, Sn$ son predicados. A este conjunto se le puede decir hipótesis o antecedente.
- S es el predicado que se infiere como verdad. Se le puede decir conclusión o consecuente.

Reglas de inferencia

- Es un método para inferir alguna verdad sobre algún predicado a partir de otros predicados.
- Se define de la siguiente manera: $\frac{S1, S2, \ldots, Sn}{S}$.
- Donde $S1, S2, \ldots, Sn$ son predicados. A este conjunto se le puede decir hipótesis o antecedente.
- S es el predicado que se infiere como verdad. Se le puede decir conclusión o consecuente.
- Entonces, ¿cómo se ve un axioma?.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1` {`a < 10`} de en forma {P}S{Q}.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1` $\{$`a < 10`$\}$ de en forma $\{P\}S\{Q\}$.
- S es la instrucción, por lo que S = `a = b / 2 - 1`.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1` {`a < 10`} de en forma {P}S{Q}.
- S es la instrucción, por lo que S = `a = b / 2 - 1`.
- {`a < 10`} es la postcondición, por lo que Q = {`a < 10`}.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1 {a < 10}` de en forma {P}S{Q}.
- S es la instrucción, por lo que S = `a = b / 2 - 1`.
- `{a < 10}` es la postcondición, por lo que Q = `{a < 10}`.
- P = ??????. Necesitamos saber algo de la variable `b` para que Q sea verdad.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1` $\{$`a < 10`$\}$ de en forma {P}S{Q}.
- S es la instrucción, por lo que S = `a = b / 2 - 1`.
- $\{$`a < 10`$\}$ es la postcondición, por lo que Q = $\{$`a < 10`$\}$.
- P = ??????. Necesitamos saber algo de la variable `b` para que Q sea verdad.
- Si reemplazamos la asignación de `a` en su postcondición entonces tenemos que `b / 2 - 1 < 10` $\Longrightarrow$ `b < 22`.

Precondiciones y postcondiciones

- Queremos expresar `a = b / 2 - 1` $\{$`a < 10`$\}$ de en forma {P}S{Q}.
- S es la instrucción, por lo que S = `a = b / 2 - 1`.
- $\{$`a < 10`$\}$ es la postcondición, por lo que Q = $\{$`a < 10`$\}$.
- P = ??????. Necesitamos saber algo de la variable `b` para que Q sea verdad.
- Si reemplazamos la asignación de `a` en su postcondición entonces tenemos que `b / 2 - 1 < 10` $\Longrightarrow$ `b < 22`.
- Por lo tanto P = $\{$`b < 22`$\}$.

Interprete mental

- ident = var
- ident = var bin-op var
- ident = un-op var
- **goto** label
- **if** var relational-op var **goto** label

C for

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

meaning

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

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.
- A thorough discussion of the use of denotational semantics to describe the semantics of programming languages is necessarily long and complex.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.
- A thorough discussion of the use of denotational semantics to describe the semantics of programming languages is necessarily long and complex.
- The idea is based on the fact that there are rigorous ways of manipulating mathematical objects but not programming language constructs.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.
- A thorough discussion of the use of denotational semantics to describe the semantics of programming languages is necessarily long and complex.
- The idea is based on the fact that there are rigorous ways of manipulating mathematical objects but not programming language constructs.

- The process of constructing a denotational semantics specification for a programming language requires one to define **for each language entity** both **a mathematical object** and a function that maps instances of that language entity onto instances of the mathematical object.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.
- A thorough discussion of the use of denotational semantics to describe the semantics of programming languages is necessarily long and complex.
- The idea is based on the fact that there are rigorous ways of manipulating mathematical objects but not programming language constructs.
- The process of constructing a denotational semantics specification for a programming language requires one to define **for each language entity** both **a mathematical object** and a function that maps instances of that language entity onto instances of the mathematical object.
- Because the objects are rigorously defined, they model the exact meaning of their corresponding entities.

Intro a Semántica Denotacional

- Denotational semantics is the most rigorous and most widely known formal method for describing the meaning of programs.
- It is solidly based on recursive function theory.
- A thorough discussion of the use of denotational semantics to describe the semantics of programming languages is necessarily long and complex.
- The idea is based on the fact that there are rigorous ways of manipulating mathematical objects but not programming language constructs.
- The process of constructing a denotational semantics specification for a programming language requires one to define **for each language entity** both **a mathematical object** and a function that maps instances of that language entity onto instances of the mathematical object.
- Because the objects are rigorously defined, they model the exact meaning of their corresponding entities.
- **The difficulty with this method** lies in creating the objects and the mapping functions.

References


Robert Sebesta (2023)
Concepts of programming languages

The End

Questions? Comments?