Introduction
CS 152 (Spring 2021)

Harvard University

Tuesday, January 26, 2021
Programming Languages

▶ More than a catalog of languages and what they can be used for.
▶ In this class: foundations of programming languages, the underlying concepts and principles that go into designing and implementing programming languages.
▶ How can you learn new languages? How can you design effective languages?
Why?

- give you the concepts to more easily learn new languages
- ... and to design and implement new languages
- golden age of PL
- elegant math
Cool: Type safety
Example: Rust

Rust is memory safe (no deferencing of null pointers, no dangling pointers), but performance is comparable to C and C++. Lots of memory checking is done statically. Achieves this using a sophisticated type system, with parametric polymorphism and linear types. All at compile time, with no run-time overhead.
Cool: Certified compilers
Cool: Program Synthesis
Cool: Program Verification
ToC

- semantics
- lambda calculus
- types
- reasoning about programs
- misc. topics
Semantics of Programming Languages

Give *mathematical meaning* to programs.
Why *mathematical*?

- Less ambiguous.
- More concise.
- Formal arguments.
Semantics
Styles of Semantics

Operational Semantics
Denotational Semantics
Axiomatic Semantics
Algebraic Semantics
Operational Semantics

Small-Step
Large-Step
step from configuration to configuration:

\[ c_0 \rightarrow c_1 \rightarrow \ldots \rightarrow c_n \]
Large-Step Operational Semantics

one step from initial configuration to final answer:

\[ c \Downarrow a \]
Denotational Semantics

interpret in mathematical domain

\[
[[\text{term}]] = \text{number}
\]

\[
[[e_1 + e_2]] = [[e_1]] + [[e_2]]
\]

\ldots
Axiomatic Semantics

\[ \{\text{Pre}\} \ c \ \{\text{Post}\} \]
Algebraic Semantics
Abstract Syntax
Abstract Syntax

\[ x, y, z \in \text{Var} \]
\[ n, m \in \text{Int} \]
\[ e \in \text{Exp} \]
Abstract Syntax

\[ x, y, z \in \text{Var} \]

\textbf{Var} is the set of program variables (e.g., foo, bar, baz, i, etc.).
Abstract Syntax

\[ n, m \in \text{Int} \]

\text{Int} is the set of constant integers (e.g., 42, \(-40\), 7).
Abstract Syntax

\[ e \in \text{Exp} \]

\textbf{Exp} is the domain of expressions, which we specify using a BNF (Backus-Naur Form) grammar.
Simple Expressions

\[ e ::= x \]
\[ \mid n \]
\[ \mid e_1 + e_2 \]
\[ \mid e_1 \times e_2 \]
Abstract Syntax Tree

\[ 1 + 2 \times 3 \]
Abstract Syntax Tree

$1 + 2 \times 3$

$1 + (2 \times 3) \quad (1 + 2) \times 3$
Abstract Syntax Tree

\[ 1 + 2 \times 3 \]

\[
\begin{array}{c}
+ \\
/ \ \\
/ \\
/ \\
1 \ * \\
/ \ \\
/ \\
2 \ 3
\end{array}
\]

\[ 1 + (2 \times 3) \quad (1 + 2) \times 3 \]
Abstract Syntax Tree

1 + 2 \times 3

\[
\begin{array}{c}
+ \\
/ \ \\ 1 \\
\end{array}
\quad \quad \begin{array}{c}
\times \\
/ \ \\ 2 \\
\end{array}
\]

1 + (2 \times 3)

\[
\begin{array}{c}
* \\
/ \ \\ + \\
/ \ \\ 1 \\
\end{array}
\quad \quad \begin{array}{c}
\times \\
/ \ \\ 3 \\
\end{array}
\]

(1 + 2) \times 3
Def. and Use of Abstract Syntax

- in OCaml
- in Coq