An interpreter (or compiler) is a program that operates on programs.
In fact, there are numerous other ways to operate on programs. For example,
- Given a one-parameter function in some language, produce the function that computes its derivative.
- Given a C program, add statements that check for memory index bounds errors.
The development of program-analysis tools of this sort is an active research area.

The Halting Problem

- For example, would be very useful to know “Is there some input to Scheme function \(P\) that will cause it to go into an infinite loop?” Is there a program that operates on programs that will answer this question correctly in finite time?
- This question was answered negatively in the 1930s by Alan Turing. In fact, there isn’t even a program that fully meets the following specification:

\[
\text{true iff DEFN is a Scheme definition that defines a one-argument function that eventually halts given the input X.} \\
\text{(define (halts? defn x) ...)}
\]

Biting Your Tail: Proof of Impossibility

\begin{verbatim}
(define (halts? defn x) alleged definition of halts?)
(define halts?-bogus-program
  (quote (define (halts?-bogus x)
    (define (halts? defn x) alleged definition of halts?)
    (define (loop) (loop))
    (if (halts? x x) (loop) #t))))
(halts? halts?-bogus-program halts?-bogus-program) ; (*)
\end{verbatim}

- Assume that \(\text{halts?}\) works as specified: \(\text{halts? defn y}\) returns true if \(\text{defn}\) is a Scheme definition of some one-argument function that halts (does not loop) when given input \(y\).
- Then if the line marked (*) returns true, it is supposed to mean that \(\text{halts?-bogus-program}\) halts.
- But \(\text{halts?-bogus}\) computes \(\text{halts? x x}\) during its execution, with the value of \(x\) being \(\text{halts?-bogus-program}\).
- That would presumably return true, which would make \(\text{halts?-bogus}\) loop infinitely.
- So clearly, if \(\text{halts?}\) works, line (*) cannot return true after all; it must return false.

Biting Your Tail (II)

\begin{verbatim}
(define (halts? defn x) alleged definition of halts?)
(define halts?-bogus-program
  (quote (define (halts?-bogus x)
    (define (halts? defn x) alleged definition of halts?)
    (define (loop) (loop))
    (if (halts? x x) (loop) #t))))
(halts? halts?-bogus-program halts?-bogus-program) ; (*)
\end{verbatim}

- But if the line marked (*) returns false, then the execution of \(\text{halts?-bogus}\) would terminate, which would mean that \(\text{halts?}\) had gotten the wrong answer.
- The only way out is to conclude that \(\text{halts?}\) never returns in this case—it does not answer the question for all possible inputs.
- Putting it all together, we must conclude that

No possible definition of \(\text{halts?}\) works all the time.

Not Just a Trick

- Nothing in this argument is specific to Scheme.
- Furthermore, Scheme is capable of representing any “effectively computable” function on symbolic data (i.e., computable via some finitely describable algorithm that terminates).
- Therefore, the impossibility of the halting problem is fundamental: the \(\text{halts?}\) function is uncomputable.
- If \(\text{halts?}\) always returns a correct result (when it returns), then there must be an infinite number of inputs for which it fails to give any answer at all (i.e., loops infinitely). Why infinite?

Consequences

- There’s a lot of fallout from the impossibility of writing \(\text{halts?}\).
- For example, I cannot tell in general whether two programs compute the same thing. [Why not?]
- Therefore, Perfect anti-virus software is theoretically impossible.
- Anti-virus software must either miss some viruses, or prevent some innocent programs from running (or freeze your computer.)
- Many analyses that might be useful cannot be done in general. For example, even if I know that a given program will terminate, I cannot necessarily predict in general how long it will take to do so.
The Mathematics of Mathematics

Gottlob Frege (1879) is usually credited with introducing the first modern formal system for expressing mathematical and logical statements and arguments. He was attempting to put mathematics on a firm foundation—to make it clear when a proof was a proof, for example.

Frege invented a universal syntax for expressing mathematical statements. Examples (with modern notation underneath):

\[
\begin{align*}
\ H(j) & \quad H(j) \\
S(s) & \quad S(s) \rightarrow M(a) \\
S(s) & \quad S(s) \rightarrow M(a)
\end{align*}
\]

\[
S(s) \rightarrow H(j) \\
S(s) \rightarrow H(j)
\]

\[
S(s) \rightarrow M(a) \\
S(s) \rightarrow M(a)
\]

\[
S(s) \rightarrow M(a) \text{ or } \exists a(P(a) \land M(a))
\]

From Syntax to Semantics

- Notations like these provide notation (syntax) without meaning (semantics), …
- …except for a few key symbols with fixed meanings:
  - Logical connectives, such as '∧', '¬', '⇒', '⇔'.
  - Quantifiers, such as '∀' (for all), '∃' (there exists), and the variables they apply to (but we don't say what set ("domain") they quantify over.)
  - (Sometimes) the predicate '∈'.
- But otherwise, the functions and predicates (true/false functions) are uninterpreted.
- So what good is it? How can we get meaningful information by just manipulating meaningless symbols?

Meaning from Assertions

- Even if we can't say exactly what a symbols means, we can assert various sentences about it that constrain its possible meanings.
- For example, suppose that, besides the standard logical connectives, quantifiers, and ¬, we allow only the relation predicate ≤.
- If we say nothing else, ≤ could mean anything.
- But suppose we assert a few things:
  \[
  ∀x, y(x = y \lor y < x) \\
  ∀x, y(x < y \land y < x \rightarrow x = y) \\
  ∀x, y, z(x \leq y \land y \leq z \rightarrow x \leq z)
  \]
  - This restricts the possible meanings of ≤ to total orderings.
  - Certain other things must now be true. E.g., ∀x(x ≤ x).
- But there are additional statements involving only ≤ whose truth is not so constrained. Example? ∃y∀x(y ≤ x)
- For our "theory of ≤", it is possible to add additional axioms to eliminate all such independent statements. Is this always possible?

Proofs

- Big Idea: If we can add enough constraints to get the properties we want for our symbols, we can dispense with messy meanings (semantics) and do everything by manipulations of syntax (e.g., which we could represent as operations on Scheme expressions).
- We call these constraining assertions
  - Axioms: (e.g., ∀x, y(x ≤ y \lor y ≤ x))
  - Axiom schemas: templates standing for an infinite number of axioms, such as A \land B → A.
- A proof of a statement, A, is defined as a finite sequence of finite statements ending with A such that each statement is either
  - An axiom (like ∀x, y(x + y = y + x), or an instance of an axiom schema (like x < y \land y < z \Rightarrow x < y, which is the result of plugging x < y and y < z into A \land B \Rightarrow A)); or
  - The result of applying one of a few inference rules to preceding statements in the proof. Most well-known inference rule is modus ponens: can add D to a proof if there are preceding statements C and C \Rightarrow D. Usually don’t have too many other rules.

Proofs (II)

- The set of axioms and schemas is finite, and a program can tell if it is looking at an axiom.
- Likewise, the inference rules must be finite and algorithmically checkable.
- Given an alleged formal proof, it is a purely clerical task to determine that it actually is a proof.
- A mathematician’s secretary or a program can make this determination.
- Furthermore, if a proof of \( A \) exists, can find it in finite (albeit enormous) time by generating and checking all possible proofs.
In fact, it allows us to write and prove theorems about whether we want them to.
Can all the true things be proven?
That we can't know whether some statements
(At which point, all statements are provable, which is
Gödel's result, however, tells us that when a system becomes
powerful enough (specifically, when it encompasses enough of the
theory of the integers), it is no longer possible to complete it in this fash-
on, except by adding contradictory axioms that make our system
inconsistent. (At which point, all statements are provable, which is
useless.)
One implication:
There must be non-standard models of arithmetic—interpretations
in which there are integers other than the familiar 0, 1, 2, ....