CSCI 334:
Principles of Programming Languages

Lecture 2: What is a language anyway?

Instructor: Dan Barowy
Williams

1

Topics

What is a language?

Turing equivalence

WCMA activity

2

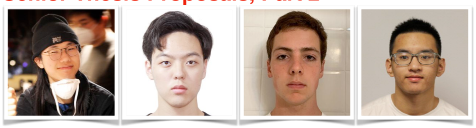
Your to-dos

1. Read *A Slightly Longer Introduction to F#*
2. Lab 1, **due Monday 2/12** (partner lab)
   Be sure to tell me who your partner is in `collaborators.txt` file.

3

Announcements

- Colloquium this **Friday, Feb 9** in Wege Auditorium at 2:35pm. **Senior Thesis Proposals, Part 2**

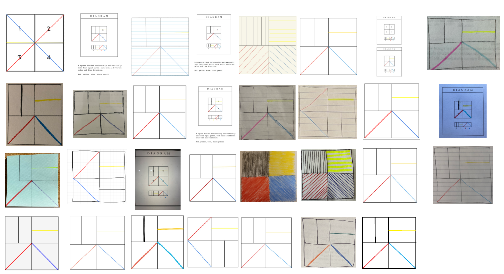


4

What is a program?

5

What makes a program a program? You might start by thinking about what you might see as output when you run a program repeatedly. Is it different every time or the same?



6

Here are all the interpretations you sent me. Although there is some variation, they are surprisingly similar.

7

It’s “supposed” to look like the image on the right. We were pretty close. So are the instructions I gave you in the homework a “program”? In a way, yes, because we all produced something similar. In another way, definitely not, because none of our outputs looked exactly like this.

(By the way, you can go view the original at MassMoCA.)

What is a language?

In this class, we concern ourselves with a specific formulation of “language,” called a **formal language**.

A **formal language** is the set of words whose letters are taken from some **alphabet** and whose construction follows some **rules**.

Example:

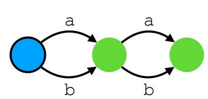
```
L = {a, aa, b, bb, ab, ba}
Σ = {a, b}
<expr> ::= <letter> | <letter><letter>
<letter> ::= a | b
```

8

L is the language: the set of all words.
Sigma is the set of symbols that comprise those words. Note that L can be very large. For English, L is infinite! But alphabets are usually finite. In fact, they are usually small.
The funny looking lines at the bottom are an example of the rules of a language, called a grammar. This grammar is written in Backus-Naur Form. We will revisit this formalism throughout the semester.

Interesting fact: language = machine

```
<expr> ::= <letter> | <letter><letter>
<letter> ::= a | b
```



The above is a machine called a deterministic finite state automaton (DFA) that “accepts” only words ∈ L.

The two definitions above describe the same language, but precisely.

9

Backus-Naur Form and finite state automata are equivalent, so you can draw them either way. For a fun example of a kind of visualization, look up Niklaus Wirth’s “syntax diagrams”, sometimes called “railroad drawings.” Importantly, both formalisms describe the fact that generating a sentence or checking that a sentence belongs to a language is a *process*, and processes can be done by machines.

What is a programming language?

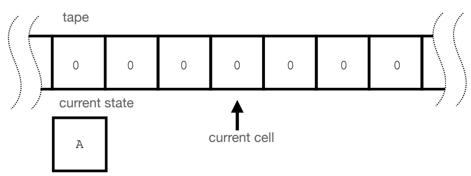
A **programming language** is defined by two machines:

1. A **syntax machine** that determines the set of strings that are in the language.
2. A **semantics machine** that determines what gets done (i.e., what computational work) with an accepted string.

We spend a lot of time in PL
thinking about these machines,
which we call **language models**.

10

Turing machine

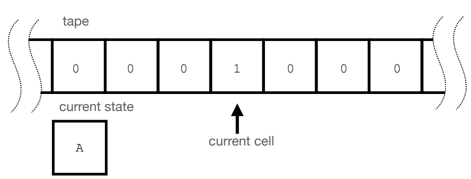


configuration			behavior		
Current state	Scanned symbol		Print symbol	Move tape	Final (i.e. next) state
A	0		1	R	B
A	1		1	L	C
B	0		1	L	A
B	1		1	R	B
C	0		1	L	B
C	1		1	N	H

11

Here is one standard language model: the Turing machine. Its operation is very simple. See if you can determine its next steps using the table below.

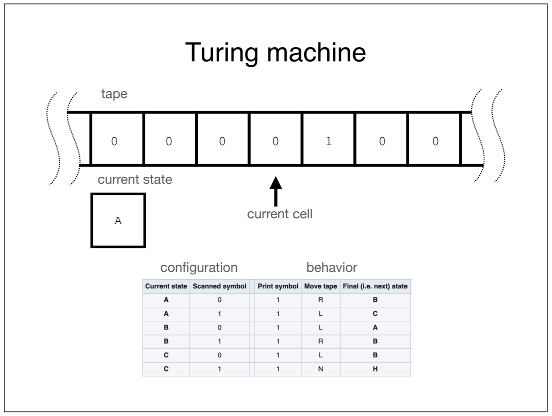
Turing machine



configuration			behavior		
Current state	Scanned symbol		Print symbol	Move tape	Final (i.e. next) state
A	0		1	R	B
A	1		1	L	C
B	0		1	L	A
B	1		1	R	B
C	0		1	L	B
C	1		1	N	H

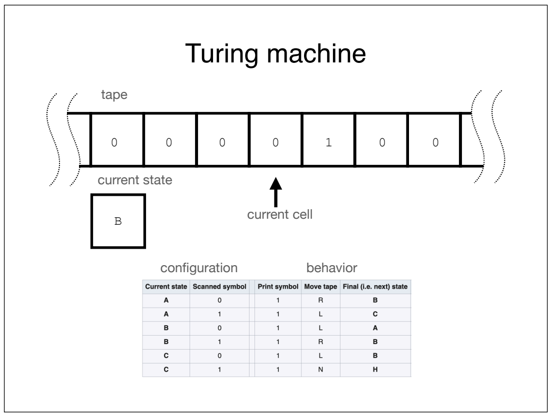
12

When in state A and current symbol 0, we write a 1 to the tape, then…



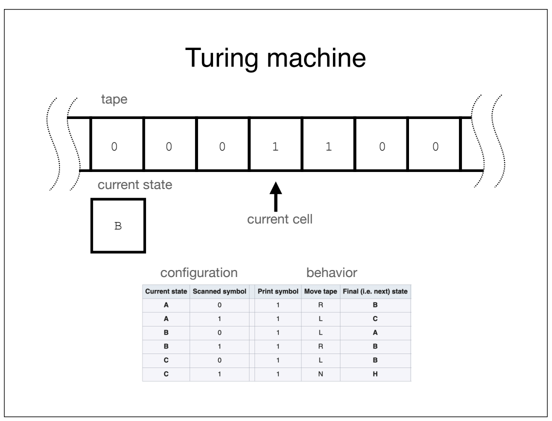
13

… move the tape to the right …



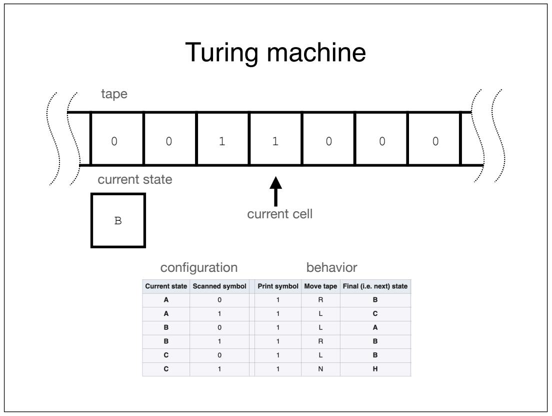
14

… and then update the current state to B.

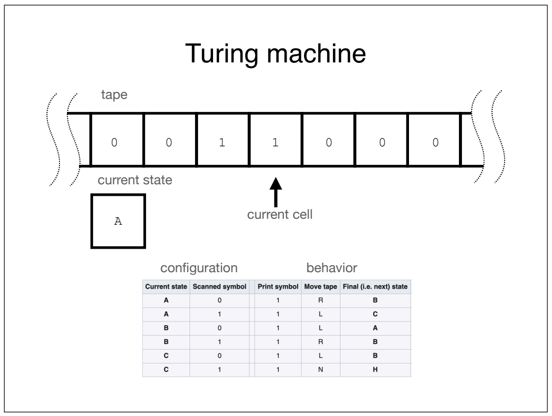


15

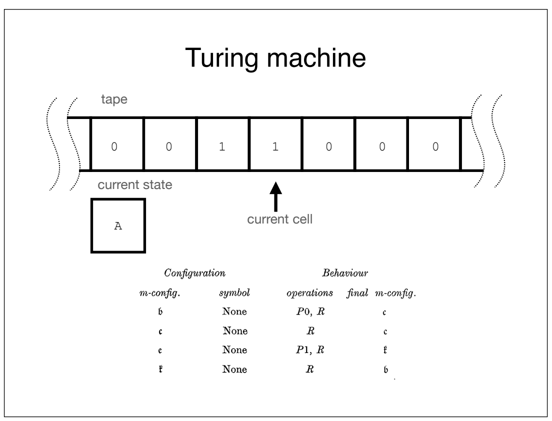
Next, with B in the current state with 0 in the current cell, we write a 1 and …



16 … move the tape to the left …



17 … and update the current state to A. And so on until the machine halts (“H”).



18 Here is the table Turing shows from his seminal paper, “On Computable Numbers, with an Application to the Entscheidungsproblem”

Surprising fact!

Almost all general purpose programming languages are **equivalent** in computational power to a **Turing machine**.

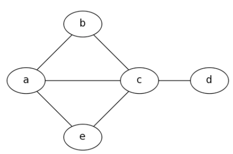
19

Believe it or not, this very simple machine is universal in the sense that all known computations can be performed on it (albeit inconveniently). That makes it good for studying lots of questions about computation. For example, in the Turing machine model, determining the cost of an algorithm is simple: assume each instruction is unit cost and count the number of instructions executed.

Domain specific languages

A **domain-specific language** (DSL) is a computer language specialized to a particular application domain. DSLs are **intentionally** not Turing equivalent, **for simplicity**.

```
graph {
    a -- b;
    b -- c;
    a -- c;
    d -- c;
    e -- c;
    e -- a;
}
```



20

But there are other models, and we don't need to use something as powerful as a Turing machine. For example, if all you want to do is to draw a graph, you might use graphViz, which is a declarative language for drawing a graph. There is a 1-to-1 correspondence between the program on the left and the image on the right. Easy to understand, and is purpose-built to draws graphs easily, but if you want to do something more sophisticated, you'll need to use a different language.

Keep in mind: two machines

21

Activity

22

Do the activity in the handout. The idea is to *describe* what you see, not to *interpret* it. Describe what you're looking at in terms a computer can understand. We like to start this activity with the Sol LeWitt wall drawing in the entrance of WCMA.

Recap & Next Class

Today we covered:

WCMA

Next class:

F#

23