Decidable Problems

Mark Greenstreet, CpSc 421, Term 1, 2006/07

- Some Relevant Hilbert Problems
  - Is mathematics complete?
  - Is mathematics consistent?
  - Is mathematics decidable?
- Decision Problems for Regular Languages and CFLs
- Some more decision problems

Hilbert and the Formalist Program

- All of mathematics can be axiomatized (e.g. Peano arithmetic, Zermelo-Fraenkel set theory).
- The notion of a proof can be formalized.
  - If C is a claim, then a proof, P, for C is a sequence of statements in the logic.
  - In these formal systems, checking that P is a valid proof for C can be done completely mechanically, much like a compiler checking a program for syntax or type-checking errors.
- This led Hilbert to propose a grand vision for mathematics.

The Hilbert Questions

- Twenty-three questions that Hilbert raised in a lecture in 1900 as being among the most important questions for mathematicians in the 20^{th} century.
- We'll focus on:
  - Is mathematics complete?
    I.e. Can any true statement be proven?
  - Is mathematics consistent?
    I.e. Is it impossible to prove a contradiction?
  - Is mathematics decidable?
    I.e. Given any claim, is there a procedure by which we can derive a proof for the claim or refute it.
- The last one, like many of Hilbert's questions, asked for a procedure. This goes back to "What is an algorithm?"

What is an Algorithm?

- Prior to Church & Turing: a description of how to compute something.
  - This seems to have been Hilbert's idea in, for example, asking for a procedure with a finite number of steps to determing whether or not a polynomial has an integral root.
  - Gauss and the FFT.
- With Church and Turing, we can be much more precise:
  - We can say what operations are allowed.
  - We can reason about the time and memory required.
  - We can show that there are problems for which no algorithm exists.
- This led to showing the impossibility of solving several of Hilbert's problems, and with it, the impossibility of completing the formalist program.

Decidable Problems Regular Language

- Decidable problems for Regular Languages
  - Does DFA M accept string w?
  - Is the language of M empty?
  - Does NFA M accept string w?
  - Does regular expression E match string w?
  - Do two DFA/NFA/REs generate the same language?
  - Just about any reasonable question you can ask about a DFA, NFA or RE.
- Decidable problems for CFLs
  - Does CFG G generate string w?
  - Does CFG G generate the empty language?

Does DFA D Accept w? (Java 1/2)

- Let $D = (Q, \Sigma, \delta, q_0, F)$.
- Describing the DFA:
  - Q: we'll just use the integers, $0 \ldots (|Q| - 1)$.
  - Σ: likewise, we'll juse the integers, $0 \ldots (|\Sigma| - 1)$.
  - δ: We'll use an array:

    ```
    int[][] delta = new int[|Q|][|Σ|] = { ...};
    ```

    We initialize `delta` so that `delta[q][c] =` $\delta(q, c)$.
  - q_0: We assign integers to states in Q so that 0 corresponds to q_0.
  - F: We'll use an array:

    ```
    boolean[] F = new boolean[|Q|] = { ...};
    ```

    We initialize `accept` so that `F[q]` is `true` iff `q` $\in F$.

Does DFA D Accept w? (Java 2/2)

```
boolean accept(int[] w){
  int q = 0; // current state
  for(int i=0; i < w.length; i++) // each symbol
    q = delta[q][w[i]]; // update state
  return(F[q]); // accept iff we reached an accepting state
}
```

Does DFA D Accept w? (TM 1/3)

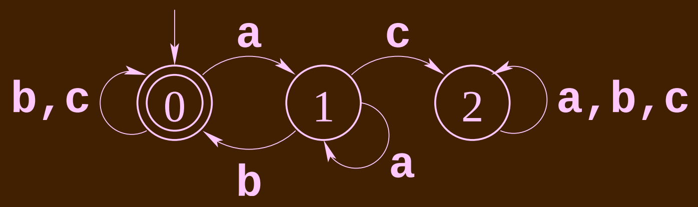
$\Sigma = \{\texttt{0}, \texttt{1}, \texttt{(}, \texttt{,}, \texttt{)}, \texttt{\#}\}$: use a binary encoding of M.

$\Gamma = \Sigma \cup \{\square, \ldots\}$

Tapes:

- Q_D: The number of states of M.
- Σ_D: The number of symbols in M's alphabet.
- δ_D: A list of tuples: $\texttt{(}\ q\ \texttt{,}\ c\ \texttt{,}\ q'\ \texttt{)}$ to indicate $\delta(q, c) = q'$.
- F: A list of accepting states – binary numbers separated by commas.
- w: The input string: binary numbers separated by commas.
- q: The current state.
- c: The current input symbol.
- $scratch$: A tape for scratch work.

Does DFA D Accept w? (TM 2/3)



The Input Tapes:

Q_D	$=$	`11`,	three states
Σ_D	$=$	`11`, three input symbols: a $\longrightarrow$ `00`, b $\longrightarrow$ `01`, c $\longrightarrow$ `10`	
δ_D	$=$	`(00,00,01),(00,01,00),(00,10,00),`	
		`(01,00,01),(01,01,00),(01,10,10),`	
		`(10,00,10),(10,1,10),(10,10,10),`	transitions
F	$=$	`00`,	the accept state
w	$=$	`00,01,00,00,01,10`,	sample input

Or, we could combine it all into one tape:

`11,11,(00,00,01),(00,01,00),(00,10,00),...`

`(10,10,10)00#00,01,00,00,01,10`$\Box^\omega$

Does DFA D accept w? (TM 3/3)

- Check that tapes Q_D, Σ_D, δ_D, and F describe a valid DFA:
- Check that tape w describes a valid input string.
- Process w:
- $\therefore$ The language $\{D\#w \mid D \text{ is a DFA that accepts } w\}$ is Turing decidable.

Does DFA D accept w? (TM 3/3)

- Check that tapes Q_D, Σ_D, δ_D, and F describe a valid DFA:
  - Make sure that δ_D has an entry for every state and input symbol (use the scratch tape as a counter). Make sure that the destination state is in $0 \ldots (|Q_D| - 1)$.
  - Make sure that every state in F is a valid state.
- Check that tape w describes a valid input string.
- Process w:
- $\therefore$ The language $\{D\#w \mid D \text{ is a DFA that accepts } w\}$ is Turing decidable.

Does DFA D accept w? (TM 3/3)

- Check that tapes Q_D, Σ_D, δ_D, and F describe a valid DFA:
- Check that tape w describes a valid input string.
- Process w:

```
q ← 0;
while more symbols in w {
  c ← the next symbol of w
    -- this moves the head for the w tape
    -- one symbol of Σ_D to the right.
  scan the δ tape to find a match for q and c.
  update q ← q'.
}
scan the F tape to find a match for q.
If a match is found, accept.
Otherwise, reject.
```

- $\therefore$ The language $\{D\#w \mid D$ is a DFA that accepts $w\}$ is Turing decidable.

Does CFG G generate w?

- Make a NTM that guesses the derivation of w and verifies it?
- How long should the derivation be?
  - Let G' be a CNF grammar for G.
  - If $w = \epsilon$, then check to see if $S_0 \rightarrow \epsilon$.
  - Otherwise, the derivation for w in G' has $2|w| - 1$ steps.
  - Note that the procedure for converting an arbitrary grammar to CNF works is an algorithm we can execute on a TM.
- $\therefore$ The language $\{G\#w \mid G \text{ is a CFG that generates } w\}$ is Turing decidable.

Hilbert's 10^{th} Problem

- Let P be a multivariable polynomial?
- Does P have a root with integer values for all of the variables?
- Solution:
  - Make a NTM that first guesses integer values for the variables.
  - Next, the NTM verifies that they are a root.
  - If they are a root, then the NTM accepts.
  - Otherwise the NTM rejects.
- No upper bound on the size of the values for the variables.
- We have reduced Hilbert's 10^{th} Problem to the Halting Problem.

Hilbert's 10^{th} Problem

- Let P be a multivariable polynomial?
- Does P have a root with integer values for all of the variables?
- Solution:
  - Make a NTM that first guesses integer values for the variables.
  - ...
- No upper bound on the size of the values for the variables.
  - The NTM may not terminate, or ...
  - It may just be writing a guessing big number for one of the variables.
  - We can't know which is the case without solving the Halting Problem.
  - $\therefore$ Hilbert's 10^{th} problem is Turing recognizable.
- We have reduced Hilbert's 10^{th} Problem to the Halting Problem.

Hilbert's 10^{th} Problem

- Let P be a multivariable polynomial?
- Does P have a root with integer values for all of the variables?
- Solution:
  - Make a NTM that first guesses integer values for the variables.
  - $\ldots$
- No upper bound on the size of the values for the variables.
- We have reduced Hilbert's 10^{th} Problem to the Halting Problem.
  - If we could solve the Halting Problem, we could solve Hilbert's 10^{th} problem.
  - In 1970, Yuri Matijasevic showed that if we could solve Hilbert's 10^{th} problem then we could solve the Halting problem.
  - $\therefore$ Hilbert's 10^{th} problem is not Turing decidable.
  - Thus, we say that the Halting Problem and Hilbert's 10^{th} problem are equivalent.
  - We'll cover this in more detail when we get to Sipser Chapter 5.

A Caution

- Let $ADD = \{x\#y\#z \mid binary(x) + binary(y) = binary(z)\}$
- Consider:

```
if(z == x+y) accept; else while(true);
```

- This program terminates iff $z = x + y$.
  - We have shown that if we can solve the Halting Problem, then we could solve the addition problem.
  - This is true, but not very interesting.
  - We can solve the addition problem whether or not we can solve the Halting Problem.

The Odd-Perfect-Number Conjecture

- A perfect number is a number that is equal to the sum of its positive, integer factors (other than itself).
  - Example: 6 = 1 + 2 + 3.
  - Example: 28 = 1 + 2 + 4 + 7 + 14.
- Conjecture: All perfect numbers are even.
- Consider:

```
i = 1;
while(true) {
  if(perfect(i)) accept;
  else i = i+1; }
```

- This program terminates iff the Odd-Perfect-Number Conjecture is false.
- We have reduce proving the Odd-Perfect-Number Conjecture to solving the Not-Halting Problem.
- We can’t possibly reduce the Non-Halting Problem to the Odd-Perfect-Number Conjecture. Why?

Reading List:

- Today: Sipser, 4.1
- Nov. 3: Sipser, 4.2
- Nov. 6: Sipser, 4.2 (cont., midterm 2 cutoff)
- Nov. 8: Sipser, 5.1
- Nov. 10: Sipser, 5.1 (cont.)
- Nov. 13: Remembrance Day (no lecture)
- Nov. 15: Midterm 2
- Nov. 17: Sipser, 5.2
- Nov. 20: Sipser, 5.3