Decidability

Mark Greenstreet, CpSc 421, Term 1, 2008/09

- 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

- Hilbert raised 10 questions in a lecture in 1900, and added 13 more when he published the list.
  - These included many of the most important questions for mathematicians in the 20^{th} century.
  - Many of these questions were aimed at making mathematics completely rigorous.
- We'll focus on his second problem which had three parts:
  - Is mathematics complete?
    I.e. Does every true statement have a proof?
  - 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 raises the question: "What is an algorithm?"

What is an Algorithm? (1/2)

- 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.
    - Gauss described the decimation-in-time FFT algorithm in a letter to another mathematician in 1805.
    - At the end of the letter, Gauss wrote (in German):

      Although this method may seem more complicated than the usual approach, I encourage you to try both methods with a 128 point transform, and you will appreciate the superiority of the method that I have described here.
    - Gauss lacked the formal notion of an algorithm, and couldn't quantify the $O(N \log N)$ vs. $O(N^2)$ complexities of the two methods.
    - James Cooley and John Tukey independelty rediscovered Gauss's algorithm 160 years later, and became famous for it.

What is an Algorithm? (2/2)

- 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? (TM 1/3)

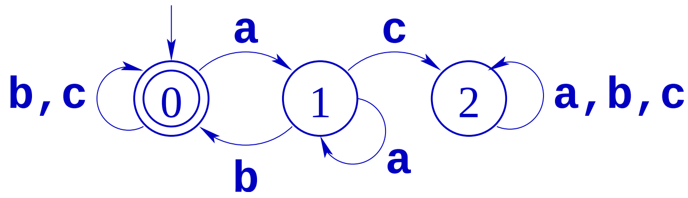
$\Sigma = \{\texttt{0}, \texttt{1}, \texttt{(}, \texttt{,}, \texttt{)}, \texttt{\#}\}$: use a binary encoding of M.

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

We'll use eight 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:

$$
\begin{aligned}
Q_D &= \texttt{11}, && \text{three states} \\
\Sigma_D &= \texttt{11}, \quad \text{three input symbols: } \texttt{a} \longrightarrow \texttt{00}, \texttt{b} \longrightarrow \texttt{01}, \texttt{c} \longrightarrow \texttt{10} && \\
\delta_D &= \texttt{(00,00,01),(00,01,00),(00,10,00),} && \\
& \texttt{(01,00,01),(01,01,00),(01,10,10),} && \\
& \texttt{(10,00,10),(10,1,10),(10,10,10),} && \text{transitions} \\
F &= \texttt{00}, && \text{the accept state} \\
w &= \texttt{00,01,00,00,01,10}, && \text{sample input}
\end{aligned}
$$

Or, we could combine it all into one tape:

$\texttt{11,11,(00,00,01),(00,01,00),(00,10,00),...}$
$\texttt{(10,10,10)00\#00,01,00,00,01,10}\square^{\omega}$

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 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.
  - Actually, we've shown this if D is written on several tapes and w is written on another one.
  - But, we could write $D\#w$ on a single input tape, and then copy it to the various tapes described above.
  - Thus, we've shown that there is a TM that decides

$$\{D\#w \mid D \text{ is a DFA that accepts } w\}$$

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 $S_0 \to \epsilon$.
  - Otherwise, the derivation for w in G' has $2|w| - 1$ steps.
  - Note that the procedure for converting an arbitrary grammar to CNF is an algorithm that we can execute on a TM.
- $\therefore$ The language $\{G\#w \mid G \text{ is a CFG that generates } w\}$ is Turing decidable.

The Halting Problem

- Let $HALT = \{M\#w \mid M$ halts when run with input $w\}$
  - M is a string that describes a TM.
  - w is a string that describes an input for M.
  - We'll give the details in later lectures.
- There is no TM that decides $HALT$.
  - I'll sketch a proof using pseudo-java programs here.
  - We'll do the mathematical proof next week.
- By the equivalence of TMs with other models of computation:
  - There is no program that can determine whether or not any give program will halt when run with any given input.
  - We'll show that just about any other property that you might want to show about what a program does is undecidable.
  - This doesn't mean that we can't prove some things about some programs.
  - It does mean that for just about any property we might be interested in, we cannot determine whether or not it holds for every program.

Halting in Java

- For the sake of contradiction, assume that we could solve the halting problem for Java programs.
- That means we could write a method: boolean halt(String p, String w) { . . .} that returns true if the program described by string p (i.e. the source code for the program) halts when run with the input given by string w.
- Now, we write the program:

```
class CounterExample;
static boolean halt(String p, String w) {
    . . .
}
public static void main(String[] args) {
    if(halt(args[0], args[0]))
        while(true);
    else System.exit(0);
}
```

- Let p be the string that is the source code for the program described above.
- What happens if we run the program, passing it p as its parameter?

java CounterExample p

- If halt(p, p) == true, then
  - The program will
    - ☐ halt
    - ☐ not halt.
  - But, halt(p, p) is supposed to mean that

    ____

- If halt(p, p) == false, then
  - The program will
    - ☐ halt
    - ☐ not halt.
  - But, $\neg$halt(p, p) is supposed to mean that

    ____

Hilbert's 10^{th} Problem

- Let P be system of polynomial equation.
- Does P have a solution with integer values for all of the variables (i.e. P is a system of Diophantine equations)?
- Solution:
  - Make a NTM that first guesses integer values for the variables, in other words, it writes the solution on a tape.
  - Next, the NTM verifies that they are a root.
  - If they are a solution, 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 system of polynomial equation.
- Does P have a solution with integer values for all of the variables (i.e. P is a system of Diophantine equations)?
- Solution:
  - ...
- 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 system of polynomial equation.
- Does P have a solution with integer values for all of the variables (i.e. P is a system of Diophantine equations)?
- Solution:
  - $\ldots$
- 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 reduced 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?

This coming week (and beyond)

- Reading
  - Today: Sipser, 4.1
  - Oct. 27 (Monday): Sipser, 4.2 (midterm cut-off)
  - Oct. 29 (Wednesday): Sipser, 4.2
  - Oct. 31 (A week from today): Sipser, 4.1
  - Nov. 3 (A week from Monday): Midterm review.
  - Nov. 5 (A week from Wednesday): Midterm 2.
- Homework
  - Today: Homework 6 due, Homework 7 goes out.
  - Oct. 31 (A week from today): Homework 7 due, Homework 8 goes out. No late homework accepted for homework 7.

    Homework 8 is extra credit.