Chapter 12: Limits of Algorithmic Computation*

Peter Cappello
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
cappello@cs.ucsb.edu

- Please read the corresponding chapter before attending this lecture.
- These notes are supplemented with figures, and material that arises during the lecture in response to questions.
- Please report any errors in these notes to `cappello@cs.ucsb.edu`. I'll fix them immediately.

*Based on **An Introduction to Formal Languages and Automata**, 3rd Ed., Peter Linz, Jones and Bartlett Publishers, Inc.

12.1 Some Problems That Cannot Be Solved by TMs

Computability & Decidability

- **Def. 9.4**: A function f with domain D is called **computable** if there is a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, such that

$$\forall w \in D \subseteq \Sigma^+,\ q_0\ w \vdash_M^* q_f\ f(w),\ q_f \in F.$$

- The "for all" part of the definition is crucial.
- Therefore, a precise statement of the *domain* is crucial.
- For decidability, the range of f is of size 2 (e.g., yes or no, true or false, 0 or 1).
- A **problem** is a set of related statements, each of which is true exclusive-or false.

- If such a problem is computable, we say it is **decidable**; otherwise, we say it is **undecidable**.

  Example: "For CFG G, $L(G)$ is ambiguous."

  The domain is the set of all CFGs.

  Example: "The Chicago Cubs will win the World Series in 2007."

  Is this a decidable problem?

- Yes: The domain has only 1 instance, which is either true or false.
- If the Cubs will win, then a TM that returns "true" correctly decides this problem;

  otherwise a TM that returns "false" correctly decides this problem.
- We do not need to prove which TM correctly decides the problem, only that one *exists*.
- Of course, Cub fans "know" which TM correctly decides *this* problem:

  The "false" TM.

Def. 12.1: Let

- w_M be a binary string that encodes a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$,
- $w \in \Sigma^+$,
- w_M and w are encoded as binary strings, as suggested in §10.4.

A solution to the **halting problem** is a TM H that, for all w_M and w, performs the computation

$$q_0\ w_M w \vdash_H^* x_1\ q_Y\ x_2,$$

if M halts on w, and

$$q_0\ w_M w \vdash_H^* y_1\ q_N\ y_2,$$

if M does not halt on w, for $q_Y, q_N \in F_H$.

Thm. 12.1: The halting problem is undecidable.
Proof: We show that there is no TM H that solves the halting problem.

1. Assume there is a TM H that solves the halting problem.
2. We require that:
   - H's input is $w_M w$
   - H halt in either q_Y or q_N appropriately (illustrate):

     $$q_0\ w_M w \vdash_H^* x_1\ q_Y\ x_2,$$

     if M halts on w, and

     $$q_0\ w_M w \vdash_H^* y_1\ q_N\ y_2,$$

     if M does not halt on w.

3. Modify H, producing H', where q_Y is *not* final (illustrate):

$$q_0\ w_M w \vdash^*_{H'} \infty,$$

if M halts on w, and

$$q_0\ w_M w \vdash^*_{H'} y_1\ q_N\ y_2,$$

if M does not halt on w.

4. Modify H', producing $\widehat{H}$, which:

   (a) copies w_M: Make M's *input* a description of *itself*

   (b) behaves like H' thereafter:

$$q_0\ w_M \vdash^*_{\widehat{H}} q_0\ w_M w_M \vdash^*_{\widehat{H}} \infty,$$

   if M halts on w_M, and

$$q_0\ w_M \vdash^*_{\widehat{H}} q_0\ w_M w_M \vdash^*_{\widehat{H}} y_1\ q_N\ y_2,$$

   if M does not halt on w_M.

5. If $\widehat{H}$'s input is a description of *itself*, then

$$q_0\ w_{\widehat{H}} \vdash^*_{\widehat{H}} \infty,$$

   if $\widehat{H}$ halts on $w_{\widehat{H}}$ (a contradiction), and

$$q_0\ w_{\widehat{H}} \vdash^*_{\widehat{H}} y_1\ q_N\ y_2,$$

   if $\widehat{H}$ does not halts on $w_{\widehat{H}}$ (a contradiction).

6. Conclusion: Our assumption that H exists is false.

The proof above is given because it uses a classic argument.
A shorter proof follows from previous results, as we now show.

Thm. 12.2: The halting problem is undecidable.
Proof:

1. Assume there is a TM H that solves the halting problem.
2. Let L be recursively enumerable but *not recursive*.
3. Then, there is a TM M such that $L = L(M)$.
4. Then, the following procedure *always halts* and accepts L:
   Given input w:
   (a) If ($H(w_M w)$ == "no") then reject & halt;
   (b) If ($M(w)$ == accept) then accept & halt; // M must halt on w.
   (c) Reject and halt.
5. Thus, L is recursive, a contradiction.
6. Conclusion: The assumption that H exists is false.

Reducing One Undecidable Problem to Another

- A problem A is **reduced** to a problem B when

$$(B \text{ is decidable }) \Rightarrow (A \text{ is decidable }).$$

- If we know that A is undecidable, and we want to show that B is undecidable, it suffices to show:

$$(B \text{ is decidable }) \Rightarrow (A \text{ is decidable }).$$

because, the contrapositive of

$$(B \text{ is decidable }) \Rightarrow (A \text{ is decidable }).$$

is

$$(A \text{ is undecidable }) \Rightarrow (B \text{ is undecidable }).$$

and since A is undecidable, by modus ponens, B is undecidable.

Example 12.1: The **state-entry problem**:
Given a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F), q \in Q$, and $w \in \Sigma^+$, decide whether or not the state q is ever entered when M is applied to w.

We show that this problem is undecidable by reducing the halting problem to it.

1. Assume we have a TM A that solves the state-entry problem.
2. If M halts, it enters a state, (p, a), from which there is no transition.
3. We use A to construct a TM H that solves the halting problem.
   - Construct M' so that (M' enters state q) $\Leftrightarrow$ (M halts on w).
     For each $(p, a) \in Q_M \times \Gamma_M$,
     If ($\delta_M(p, a)$ is defined) then set $\delta_{M'}(p, a) = \delta_M(p, a)$;
     else $\delta_{M'}(p, a) = (\mathbf{q}, a, R)$, for a new state $\mathbf{q} \in Q_{M'}$.

4. When H is given M and w, it invokes the following procedure:
   (a) Construct M' from M;
   (b) Invoke A with input M', $\mathbf{q}$, and w;
   (c) If A returns "yes", then H returns "yes"; // M halts on w
   Else return "no"; // M does not halt on w.
5. But, the halting problem is undecidable.
6. Conclusion: The assumption that TM A exists is false.

Example 12.2: The **blank-tape halting problem**:
Given a TM M, decide whether or not M halts on a blank tape.

- We reduce the halting problem to the blank-tape halting problem.
- Assume that a TM A solves the blank tape problem.
- Construct a TM H that solves the halting problem (illustrate):
  1. Given (M, w), construct M_w that, given a blank tape:
     (a) writes w onto the tape;
     (b) puts itself in the configuration $q_0\ w$;
     (c) behaves like M thereafter.
  2. If ($A(M_w)$ == "yes") then return "yes"; // M halts on w
     Else return "no"; // M does not halt on w
- But, the halting problem is undecidable.
- Conclusion: Our assumption that A exists is false.

Example 12.3: Let $f(n)$ be the maximum number of moves that can be made by any *halting* n-state TM with $\Gamma = \{0, 1, \Box\}$, when started on a *blank* tape.

- $|Q| = n$.
- $|\Gamma| = 3$.
- Thus, the number of distinct δ functions—TMs—is finite.
- Only *some* of these halt on a blank tape.
- $f(n)$: the largest number of moves made by these TMs on a blank tape.

- Reduce the blank tape halting problem to the computation of f.
- Assume that a TM M_f computes $f(n)$.
- Construct a TM that solves the blank-tape halting problem (illustrate):
  1. Given M, invoke M_f with $|Q_M|$, computing $m = f(|Q_M|)$.
  2. Construct TM $C_{M,m}$ such that:
     - It simulates M on a blank tape for at most m moves;
     - If (M halts in $\leq m$ moves) then return true;
       else return false;
  3. If ($C_{M,m}$) then return true; // M halts on a blank tape
     Else return false; // M does not halt on a blank tape
- But, the blank-tape halting problem is undecidable.
- Conclusion: Our assumption that $f(n)$ is computable is false.