Reductions Redux

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

- Reductions in Java

- A Hierarchy of hard problems
A language is a set of strings. Let \( A \) be a language.

A Java method that takes a string as an argument and returns a boolean can be a decider or a recognizer for a language.

- If the method returns \texttt{true} for every string in the language and returns \texttt{false} for every string that is not in the language, then that method is a \textit{decider}.

- If the method returns \texttt{true} for every string in the language and returns \texttt{false} or “loops” for every string that is not in the language, then that method is a \textit{recognizer}.
Previous Results, \textit{HALT}

From previous lectures, we know that:

- There is no TM that decides $\text{HALT}$ where

\[
\text{HALT} = \{ M \# w \mid \text{M is a TM that halt when run with input } w \}
\]

- By the equivalence of Java programs and TMs, we conclude that there is no Java method that decides $\text{HALT}$ (or any other programming language).

- Furthermore, we can define

\[
\text{HALT}_J = \{ J \# s \mid \text{J is a Java program that halts when run with input } s \}
\]

For simplicity, we’ll assume that the “input” to a Java program is a string. Note that we can take any collection of parameters of any types and represent them with strings.

- $\text{HALT}_J$ cannot be decided by any TM or Java program (or any program in any other language).

- $\text{HALT}$ and $\text{HALT}_J$ can be recognized by a TM or a Java program.
Previous Results, $A_{TM}$ and $E_{TM}$

$A_{TM} = \{ M \# w | \text{TM } M \text{ accepts string } w \}$.  
- $A_{TM}$ is Turing and Java recognizable but is neither Turing nor Java decidable.
- We can define $A_{Java}$ in the obvious manner, and it is Turing and Java recognizable, but neither Turing nor Java decidable.

$E_{TM} = \{ M | L(M) = \emptyset \}$.  
- $E_{TM}$ is Turing and Java co-recognizable, but is neither Turing nor Java decidable.
- This means that there is a TM (equivalently Java program) that rejects every string that is not in $E_{TM}$ and for any string in $E_{TM}$ either accepts or loops.
- We can define $E_{Java}$ in the obvious manner, and it is Turing and Java co-recognizable, but neither Turing nor Java decidable.
Some handy Java methods

I'll assume that we have the following Java methods available:

```java
String[] getArgs(String s) {
    /* return an array, args, such that
    * s = args[0] + '#' + args[1] + ...
    *   '#' + args[args.length-1]
    * and none of the args strings contain the '#' character.
    */
    return new String[0];
}

boolean simulate(String J, String s) {
    /* Simulate java program J running with input s. */
    /* If J not a valid program, return(false). */
    /* Else if J accepts s, return true. */
    /* Else if J rejects s, return false. */
    /* Else (J loops on s) never return. */
    return false;
}

boolean simulate(String s) {
    String[] args = getArgs(s);
    if(args.length != 2) return(false);
    return(simulate(args[0], args[1]));
}
```
Some more handy methods

```java
boolean anbn(String s) {
    // return true if there is an integer, n, such that s = a^n b^n.
    ...
}
```
**REGULAR and Java**

- \( \text{REGULAR} = \{ M \mid L(M) \text{ is regular, } M \text{ describes a TM} \} \).
- \( \text{REGULAR}_J = \{ J \mid L(J) \text{ is regular, } J \text{ is the source code of a Java program} \} \).
- Reducing \( A_J \) to \( \text{REGULAR} \) (using Java)
  - Assume we have a method
    ```java
    boolean regularJ(String J) { ... }
    ```
    That decides language \( \text{REGULAR}_J \).
  - We use \text{regular} to write a Java method that decides \( J \).
    ```java
    boolean aJ(String s) { /* return true if } \text{s} = J\#\text{w} \text{ and } J \text{ accepts } \text{w} */
        return(regular(
            "boolean foo(String x) {"+
            + "    if(anbn(x))"+
            + "        return(true);"+
            + "    else"+
            + "        return(simulate(" + s + "));"+
            + "})"
        )); }
    ```
**\textit{REGULAR}_J \ (\text{cont})**

- From previous slide

```java
boolean aJ(String s) {
    /* return true if s = J#w and J accepts w */
    return(regular(
        "boolean foo(String x) {
            if(anbn(x))"
        + "    return(true);
        + "    else"
        + "    return(simulate(" + s + "));
        + "})"
    ));
}
```

- If \(s\) is a string of the form \(J\#w\) and Java program \(J\) accepts input \(w\), then \(\text{foo}\) accepts all strings. Otherwise, \(\text{foo}\) only accepts strings of the form \(a^n b^n\).

- In other words, the language of \(\text{foo}\) is regular iff \(J\) accepts \(w\).

- If we could decide \(\text{REGULAR}_J\), we could also decide \(A_J\).

- \(A_J\) is not decidable (just like \(A_{TM}\)). Therefore \(\text{REGULAR}\) is not decidable either.
REGULAR is not decidable (TM-1)

- If REGULAR were decidable, then there would be a TM, $M_{REG}$ that decides it.
- We'll show that if we had $M_{REG}$, we could build another TM, $M_{ATM}$ that would decide $A_{TM}$.
- When run with input string $s$, here's what $M_{ATM}$ will do:
  - Compute the description of a TM, $M_{foo}$:
    - If run with input $x$, $M_{foo}$ will:
      - Check to see if $x$ has the form $a^nb^n$ and if so accept.
      - Otherwise, $M_{foo}$ simulates $M$ running with input $w$.
        - If $M$ accepts $w$, then $M_{foo}$ accepts $x$.
        - If $M$ rejects $w$, then $M_{foo}$ rejects $x$.
        - If $M$ loops on $w$, then $M_{foo}$ loops on $x$.
    - Note that $L(M_{foo})$ is regular iff $M$ accepts $w$.
  - $M_{ATM}$ now runs $M_{REG}$ on the description of $M_{foo}$.
    - If $M_{REG}$ accepts $M_{foo}$ then $M_{ATM}$ accepts $s$ (i.e. $M \#w$).
    - If $M_{REG}$ rejects $M_{foo}$ then $M_{ATM}$ rejects $s$.
    - If $M_{REG}$ cannot loop on $M_{foo}$ because it was assumed to be a decider.
REGULAR is not decidable (TM-2)

- If we had a TM, $M_{REG}$ that was a decider for the language $REGULAR$,
- Then we could construct a TM, $M_{ATM}$ that would be a decider for $ATM$.
- We know that $ATM$ is not decidable.
- Thus, we cannot build a decider for $REGULAR$.
- Therefore, $REGULAR$ is not Turing decidable.
Reducing $\overline{A_{TM}}$ to REGULAR

- This time, our $M_{ATM}$ will compute the description of $M_{bar}$.
  - $M_{bar}$ simulates $M$ running with input $w$.
    - If $M$ accepts $w$, then $M_{bar}$ checks to see if $x$ has the form $a^n b^n$.
      - If $x$ has the form $a^n b^n$, $M_{bar}$ accepts $x$.
      - Otherwise, $M_{bar}$ rejects $x$.
    - If $M$ rejects $w$, then $M_{bar}$ rejects $x$.
    - If $M$ loops on $w$, then $M_{bar}$ loops on $x$.

$L(M_{bar})$ is regular iff $M$ rejects $w$.

- If we had a TM, $M_{REG}$ that was a decider for the language $REGULAR$,
  - Then we could construct a TM, $M_{ATM}$ that would be a decider for $\overline{ATM}$.
  - We know that $\overline{ATM}$ is not decidable.
  - Thus, we cannot build a decider for $REGULAR$.
  - Therefore, $REGULAR$ is not Turing decidable.
This time, we write

```java
boolean aJbar(String s) { /* return true s = J#w and J rejects w */
    return(regular(
        "boolean bar(String x) {
        + " if(simulate(" + s + "))
        + " if(anbn(x)) return(true);
        + " else return(false);
        + " else return(false);
        + "$
    )
    
    If s is a string of the form J#w and Java program J accepts input w, then bar accepts strings of the form a^n b^n. Otherwise, bar rejects or loops on all strings.

    In other words, the language of bar is regular iff J does not accept w.

    If we could decide REGULAR_J, we could also decide \( \overline{A_J} \) which is equivalent to \( \overline{A_{TM}} \).

    \( \overline{A_{TM}} \) is not decidable. Therefore REGULAR is not decidable either.
How hard is $REGULAR$?

- We cannot reduce $REGULAR$ to $A_{TM}$. Why not?
  - If we could, then we could reduce $\overline{A_{TM}}$ to $A_{TM}$ – we’ve shown that we can reduce $\overline{A_{TM}}$ to $REGULAR$.
  - Then, we could build a decider for $A_{TM}$:
    Given an input $M \# w$, we could run a recognizer for $A_{TM}$ and a recognizer for $\overline{A_{TM}}$ until one accepts. If the recognizer for $A_{TM}$ accepts, we accept, and if the recognizer for $\overline{A_{TM}}$ accepts, then we reject.
  - But, $A_{TM}$ is not decidable.
  - Therefore, we can’t reduce $REGULAR$ to $A_{TM}$.

- We cannot reduce $REGULAR$ to $\overline{A_{TM}}$ either.
  The proof has the same form as the proof above.
Quantifying Decidability

- **<ExtraCredit>**

- A language, $A$, is **Turing decidable** iff there is a TM that decides it.
  - Examples: any regular or context free language, testing for primality, any NP-complete problem, anything for which you have an algorithm.

- A language, $B$, is **Turing recognizable** iff there is a Turing decidable language $A$ such that:
  \[ B = \{ s \mid \exists x. \ s \uparrow x \in A \} \]

  - Example, $HALT$. Let
    \[
    A = \{ M \# w \uparrow n \mid M \text{ describes a Turing machine, } w \text{ describes a string, and } n \text{ is the binary representation of an integer, such that TM } M \text{ halts within } n \text{ steps when run with input } w. \}
    \]

    $A$ is decidable (see midterm 2). Thus, $HALT$ is Turing recognizable.

  - In this case, we used the existential quantifier to say that if $M$ accepts $w$, then there must be some integer $n$ such that $M$ accepts $w$ after at most $n$ steps. This can be verified by simulating $M$ for at most $n$ steps.
Quantifying Decidability

Let $\text{accept}(M, w, n)$ denote that TM $M$ accepts $w$ after at most $n$ steps.

A language, $E$, is Turing co-recognizable iff there is a Turing decidable language $A$ such that:

$$B = \{ s \mid \forall x. s \uparrow x \in A \}$$

Examples

$$\overline{A_{TM}} = \{ M \# w \mid \neg \exists n. \text{accept}(M, w, n) \}$$
$$= \{ M \# w \mid \forall n. \neg \text{accept}(M, w, n) \}$$

$$\overline{E_{TM}} = \{ M \mid \forall w, n. \neg \text{accept}(M, w, n) \}$$

What about $\text{REGULAR}$?

$$\text{REGULAR} = \{ M \# w \mid \exists D. \forall w.
\text{DFArecognize}(D, w) \Rightarrow \exists n. \text{accept}(M, w, n)
\wedge \neg \text{DFArecognize}(D, w) \Rightarrow \forall n. \neg \text{accept}(M, w, n) \}$$
The Arithmetic Hierarchy

\[ \Sigma_1 = \exists x_1. p(s, x_1) \quad \text{E.g. } HALT, A_{TM} \]

\[ \Pi_1 = \forall x_1. p(s, x_1) \quad \text{E.g. } E_{TM} \]

\[ \Sigma_2 = \exists x_2. \forall x_1. p(s, x_1, x_2) \quad \text{E.g. } FINITE \]

\[ \Pi_2 = \forall x_2. \exists x_1. p(s, x_1, x_2) \quad \text{E.g. } TOTAL \]

\[ \Sigma_3 = \exists x_3. \forall x_2. \exists x_1. p(s, x_1, x_2, x_3) \quad \text{E.g. } REGULAR \]

\[ \Pi_3 = \forall x_3. \exists x_2. \forall x_1. p(s, x_1, x_2, x_3) \]

Decidable: \( p(s) \)
This coming week (and beyond)

- **Reading**
  - Today: Sipser 5.1
  - Nov. 10 (Monday): Sipser 5.2
  - Nov. 12 (Wednesday): Sipser 5.3
  - Nov. 14 (A week from today): Sipser 7.1

- **Homework**
  - Today: HW 9 goes out.
  - Nov. 10 (Monday): HW 8 due.
  - Nov. 14 (a week from today): HW 10 goes out.
  - Nov. 17 (a week from Monday): HW 9 due.