Reductions

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

- The Idea of Using Reductions
- Proving Undecidable Problems Using Reductions

Reductions

Let's say we want to solve problem of class A and we know how to solve problems of class B.

- If for every problem of class A, we can find a way to convert it into some problem of class B, . . .
- then, we can solve all problems of class A using our method for solving problems of class B.
- We can also talk about how much effort is need to transform the problem. For most of what we are interested in here, it is enough that the transformation can be computed by a Turing machine.

Reducing Multiplication to Addition

- We can convert the problem of multiplying natural numbers into the problem of addition:

  ```
  product = 0;
  for(int i = 0; i < x; i++)
    product = product + y;
  ```

  We have reduced multiplication to addition.

- We can do better if we allow bit shifts and tests:

  ```
  product = 0;
  while(x > 0) {
    if(odd(x)) product = product + y;
    x = x >> 1; y = y << 1;
  }
  ```

  We have reduced multiplication to addition and bit shifts and tests.

- How about if we have addition, right shifts, and squaring?

  ```
  product = ((x+y)ˆ2 - (x-y)ˆ2) >> 2;
  ```

More Examples

- Scheduling problems that are linear programs.
- Routing problems that are shortest path in a graph.
- Some problems that look NP complete are bipartite matching in disguise.
- NP completeness proofs are often done by reduction.
- The whole idea of programming with an API is the practical use of reductions: reducing parts of a software project to functionality that is already present in the API.

A Warning

- We can show that B is at least as hard as A by reducing A to B.
- Reducing A to B shows that A is at least as easy as B.
- Let $A = \{w \mid w \text{ is the binary representation of a composite number}\}$. We can reduce A to the halting problem:

```
i = 2;
while(true) {
  if((i != w) && (w % i) == 0) accept();
  i = i+1;
}
```

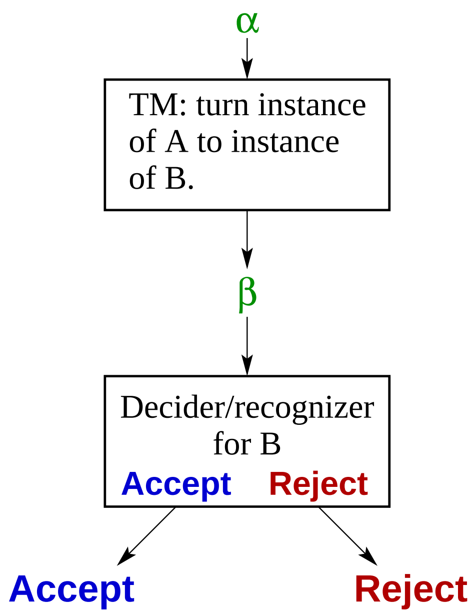
  This program halts iff w is composite. Thus, we have shown that testing for compositeness in no harder than the halting problem. In fact, it is much easier.

The Halting Problem (again)

- Let $HALT = \{M\#w \mid \text{Turing machine } M \text{ halts when run with input } w\}$.
- We can reduce A_{TM} to $HALT$:
  - Create a Turing machine N that on input $M\#w$
  - N creates a string $M'\#w$ where M' is like M but has a new state, $loop$.
  - All transitions of M to state $reject$ are replaced with transitions to $loop$.
    - If M accepts w so does M'.
    - If M rejects or loops on w, M' loops.
    - Thus, $M\#w$ in A_{TM} iff $M'\#w \in HALT$.
  - N now runs $HALT$ on $M'\#w$.
    - If $HALT$ accepts, N accepts.
    - If $HALT$ rejects, N rejects.
  - N recognizes A_{TM}.
- This shows that $HALT$ is at least as hard as A_{TM}.
- A_{TM} is undecidable, therefore $HALT$ is undecidable.
- We can show that $HALT$ reduces to A_{TM}; thus $HALT$ and A_{TM} are equivalent in hardness.

A Recipe For Reduction

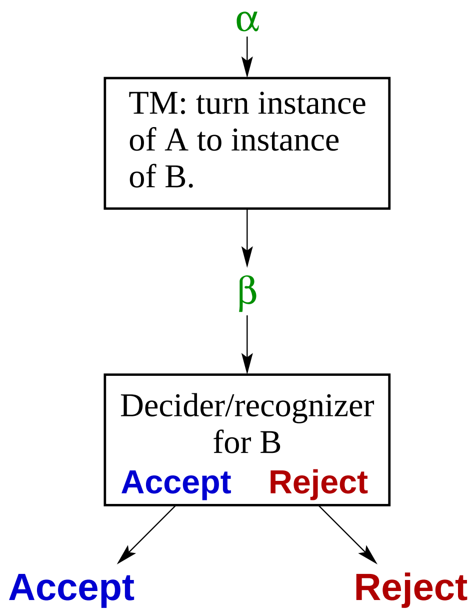
Want to show: $B \succeq A$ (B is at least as hard as A).



- We show that we can translate any question of the form "$\alpha \in A$?" to a question of the form "$\beta \in B$?".
  - Given a string α we compute a string β.
  - If this transformation can be done by a TM in a finite number of steps, then we say that it is "Turing reducible."
- Typically, A is a question about the languages or strings that a TM decides or recognizes.
  - This is because we have shown some undecidable problems for TMs (e.g. $HALT$ and A_{TM}).
  - Thus, α is often of the form M or $M\#w$ where M is the description of a TM, and w is the description of an input string to M.

A Recipe For Reduction

Want to show: $B \succeq A$ (B is at least as hard as A).



- B may be a question about the language the behaviour of TMs or some other model.
  - B is the language that we want to show is hard to decide and/or recognize?
  - Thus, β is often of the form N or $N\#x$ where N is the description of a TM, a 2-PDA, a ray automaton, $\ldots$, and x is the description of an input string to N.
- Rules for reductions
  - To show that B is at least as hard as A, reduce problems of A to problems of B.
  - Only get to “call” B once.
  - B gives the answer – you can’t invert it, etc.

Language Emptiness

- Let $E_{TM} = \{M \mid L(M) = \emptyset\}$.
- We can reduce $\overline{A_{TM}}$ to E_{TM}:
  - Create a Turing machine N that on input $M\#w$
  - N writes the description for TM M':
    - M' rejects if its input is not equal to w.
    - Otherwise, M' runs M on input w:
      - If M accepts w so does M'.
      - If M rejects w so does M'.
      - If M loops on w so does M'.
    - If M accepts w, then $L(M') = \{w\}$.
    - Otherwise $L(M') = \emptyset$.
  - N runs the machine for E_{TM} on M'.
    - If $M' \in E_{TM}$, N accepts.
    - Otherwise, N rejects.
  - $L(N) = \overline{A_{TM}}$.
- This shows that E_{TM} is at least as hard as $\overline{A_{TM}}$.

  $\therefore E_{TM}$ is undecidable.

A Note on the Proof

- We just showed that E_{TM} is at least as hard as $\overline{A_{TM}}$.
- At each step, we were careful to make sure that the machine that called the "sub-machine" would do the same thing (accept, reject, or loop) as the "sub-machine".
- If we flip accept and reject, then what should we do with loop?
- Sipser avoids this by using reduction to prove undecidability – he shows that no decider exists for the specified problem. Thus, he doesn't need to consider looping behaviours.
- Our argument shows a bit more, we've not only shown that E_{TM} is undecidable, we've also shown that it is at least as hard as Turing co-recognizable (but undecidable).
- In fact, E_{TM} is Turing co-recognizable.
- We cannot reduce a Turing recognizable (but undecidable) language to a Turing co-recognizable language. If so, we would have shown that all Turing recognizable languages are Turing co-recognizable, and this would make them Turing decidable. But, we know that there are Turing recognizable languages that are undecidable.

Anatomy of a Reduction Proof

Want to show that $\mathcal{A} \prec \mathcal{B}$.

- Let A be a language in class $\mathcal{A}$. Let w be a string.
- Find a langauge $B \in \mathcal{B}$ and construct a string w' s.t. $(w \in A) \Leftrightarrow (w' \in B)$.
- Typically, this involves a bunch of Turing Machines:
  - M_B a machine that decides (recognizes, etc.) B.
  - Often, A is defined for strings that include TM descriptions. e.g.

$$A = \{M\#w \mid \text{where } M \text{ is a TM that} \ldots\}$$

  - We define a TM, M_A that decides (recognizes, etc.) A by:
    - Constructing a new machine, M' based on M and possibly w.
    - Runs M_B on an input that includes a description of M'.
    - M_A accepts if M_B accepts and M_A rejects if M_B rejects (and loops if M_B loops).
    - NOTE: we never actually run M' on anything!

Reducing E_{TM} to $\overline{A_{TM}}$

- Let M be a Turing machine. To determing if $L(M) = \emptyset$:
- Construct a new Turing machine M'. Here's what M' does:

```
n = 1;
while(true) {
  for(i = 1; i < n; i++) {
    w = string(i);
    simulate $M$ for i steps on input w;
    if($M$ accepts) accept;
  }
}
```

- For any w, test $M'\#w \in \overline{A_{TM}}$.
  - $(M'\#w \in \overline{A_{TM}}) \Leftrightarrow (L(M') = \emptyset)$.
  - Thus, we've reduced E_{TM} to $\overline{A_{TM}}$.
- We've shown that E_{TM} is at least as hard as $\overline{A_{TM}}$ (slide 8), and that E_{TM} is at most as hard as $\overline{A_{TM}}$ (this slide).
- $\therefore E_{TM}$ is undecidable and Turing co-recognizable.

$REGULAR$ is Undecidable

- Let $REGULAR = \{M \mid L(M) \text{ is regular}\}$.
- $REGULAR$ is undecidable. On input $M\#w$ where M describes a TM and w is describes an input string for that TM
  - ...

Diagrams of Decidability

- Venn Diagrams for the sets of Turing decidable, recognizable, and co-recognizable languages.
- Describing these with quantifiers.

This coming week (and beyond)

- Reading
  - Today: Sipser 5.1
  - Nov. 3 (Monday): Midterm review.
  - Nov. 5 (Wednesday): Midterm 2.
  - Nov. 5 (A week from Today): Sipser 5.2
- Homework
  - Oct. 31 (Friday): Homework 7 due, Homework 8 goes out.
    No late homework accepted for homework 7.

    Homework 8 is extra credit.