CS F364
Design & Analysis of Algorithms

ANALYSIS – PROBLEMS – REDUCTIONS

Analysis of Problems

      - Reduction

      - Karp Reduction, 1-1 Reduction

      - Lower Bounds and Reduction: Example

      - Turing Reduction

PROBLEMS - REDUCTION

- Reduction is a mechanism for capturing the relation "at least as difficult as" between problems.

  - "at least as difficult as" refers to "at least as difficult to solve as"

    - Typically this would mean

      "**requires at least as much time to solve as**" or

      "**requires at least as much space to solve as**"

      (*depending on the context*).

PROBLEMS - REDUCTION

- Definition:

  - Let π_1 and π_2 be decision problems with input sets $I(\pi_1)$ and $I(\pi_2)$ respectively.

  - We say, π_1 ***reduces to*** π_2 ,

    - if there is a function f : $I(\pi_1)$ --> $I(\pi_2)$ such that

    - for every x $\in$ $I(\pi_1)$,

   - $\pi_1(x) = 1$ if and only if $\pi_2(f(x)) = 1$

- Questions:

1. What does it mean "algorithmically"?

2. What is the cost of reduction?

   i.e. what is the cost of (computing) the function f?

KARP REDUCTION

- Definition:

  - Let π_1 and π_2 be decision problems with input sets $I(\pi_1)$ and $I(\pi_2)$ respectively.

  - We say, π_1 **reduces to** π_2 (denoted $\pi_1 \precsim \pi_2$)
    - if there is a function $f : I(\pi_1) \dashrightarrow I(\pi_2)$ such that
    - for every $x \in I(\pi_1)$,
   - $\pi_1(x) = 1$ if and only if $\pi_2(f(x)) = 1$

Denotes an oracle for π_2

x in $I(\pi_1)$ → f → $f(x)$ → π_2 → 0/1
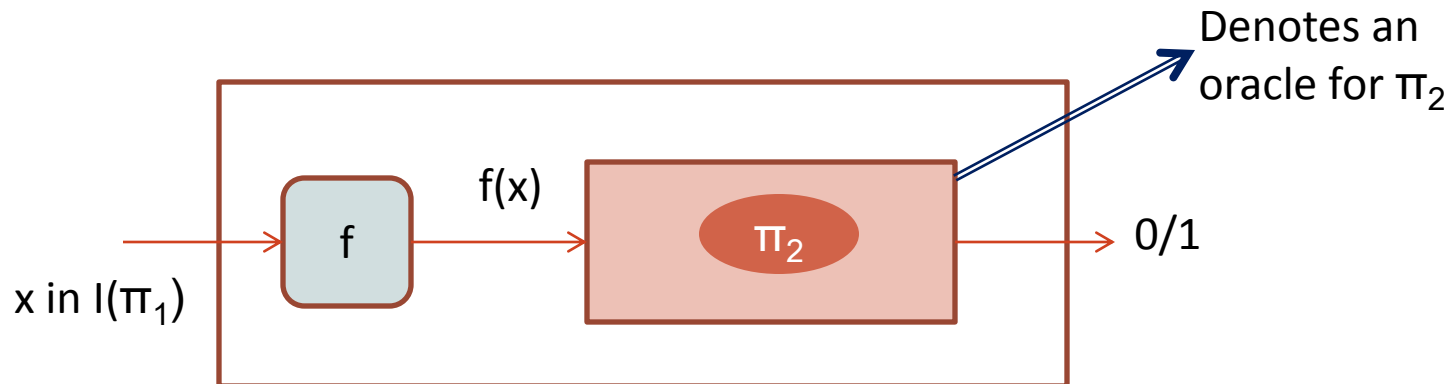
1-1 REDUCTION

- Karp reduction is a special case of 1-1 reduction:
  - Let π_1 and π_2 be problems with input sets $I(\pi_1)$ and $I(\pi_2)$ respectively.
  - We say, π_1 *reduces to* π_2 (denoted $\pi_1 \preceq \pi_2$)
    - if one can obtain an algorithm for π_1 given an algorithm for π_2
    - using mapping functions **f** and **g** on inputs to π_1 and outputs of π_2 respectively:

x in $I(\pi_1)$ → f → f(x) → [π_2] → $\pi_2(f(x))$ → g → $\pi_1(x)$
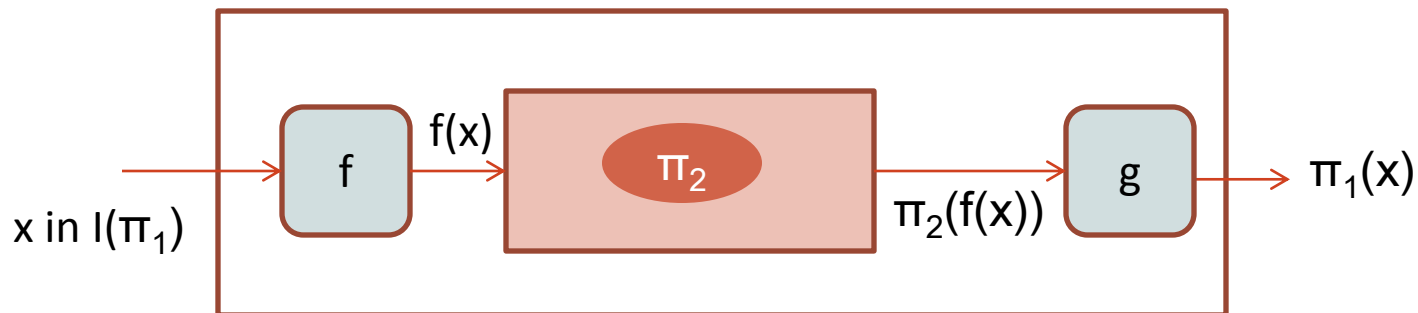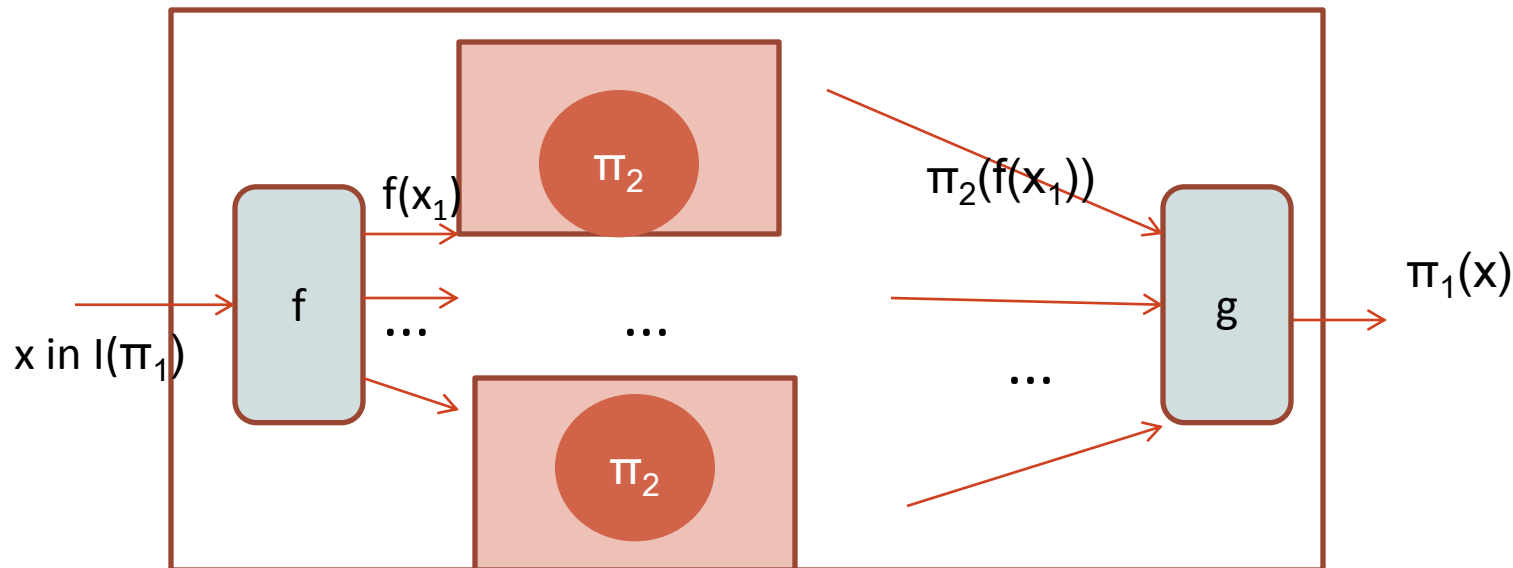
PROBLEMS – REDUCTION

- We are usually interested in efficient reductions:
  - i.e. the function mapping inputs to inputs should be "efficiently" computable
  - e.g. **f** should be computable in polynomial time.
- We use $\pi_1 \preceq_{t(n)} \pi_2$
  - to denote that π_1 ***reduces to*** π_2 in time t(n)
    - t(n) is the total cost of functions f and g.
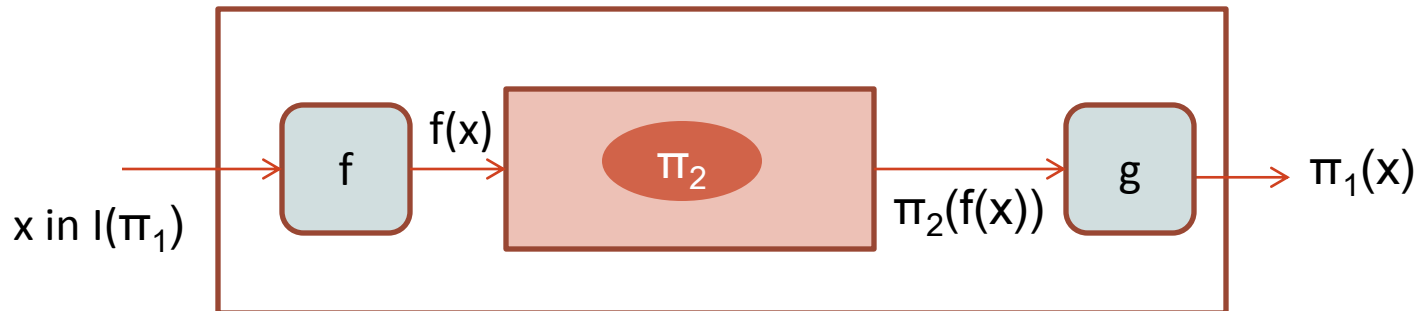
TURING REDUCTIONS

○ We use $\pi_1 \preccurlyeq_{T,t(n)} \pi_2$

- to denote that π_1 **_Turing-reduces to_** π_2 in time t(n)

  ○ t(n) is the total cost of functions f and g and the cost of calls to π_2

Implications: ???