Thinking About Algorithms

The Heart of Computer Science

Example Problems

- The Stable Marriage Problem

  Specification, Algorithm Correctness, Iteration

- Root Finding

  Top-Down Refinement, Recursion

- Greatest Common Divisor

  Time Complexity, Recursion and Iteration

- The Halting Problem

  Limits of Computation, Reduction

The Stable Marriage Problem

Input:

- The names of *n* men and *n* women.
- Each person provides a rank ordered list of everyone of the opposite sex (first to last choice).

Output: A set of (man, woman) pairs such that

- Each person is in exactly one pair.
- The arrangement is *stable*.

What do we mean by *stable?*

We say that an arrangement is *stable* if:

There does *not* exist a man *M* and woman *W such that*

- *M* prefers *W* over his own wife, and
- *W* prefers *M* over her own husband.

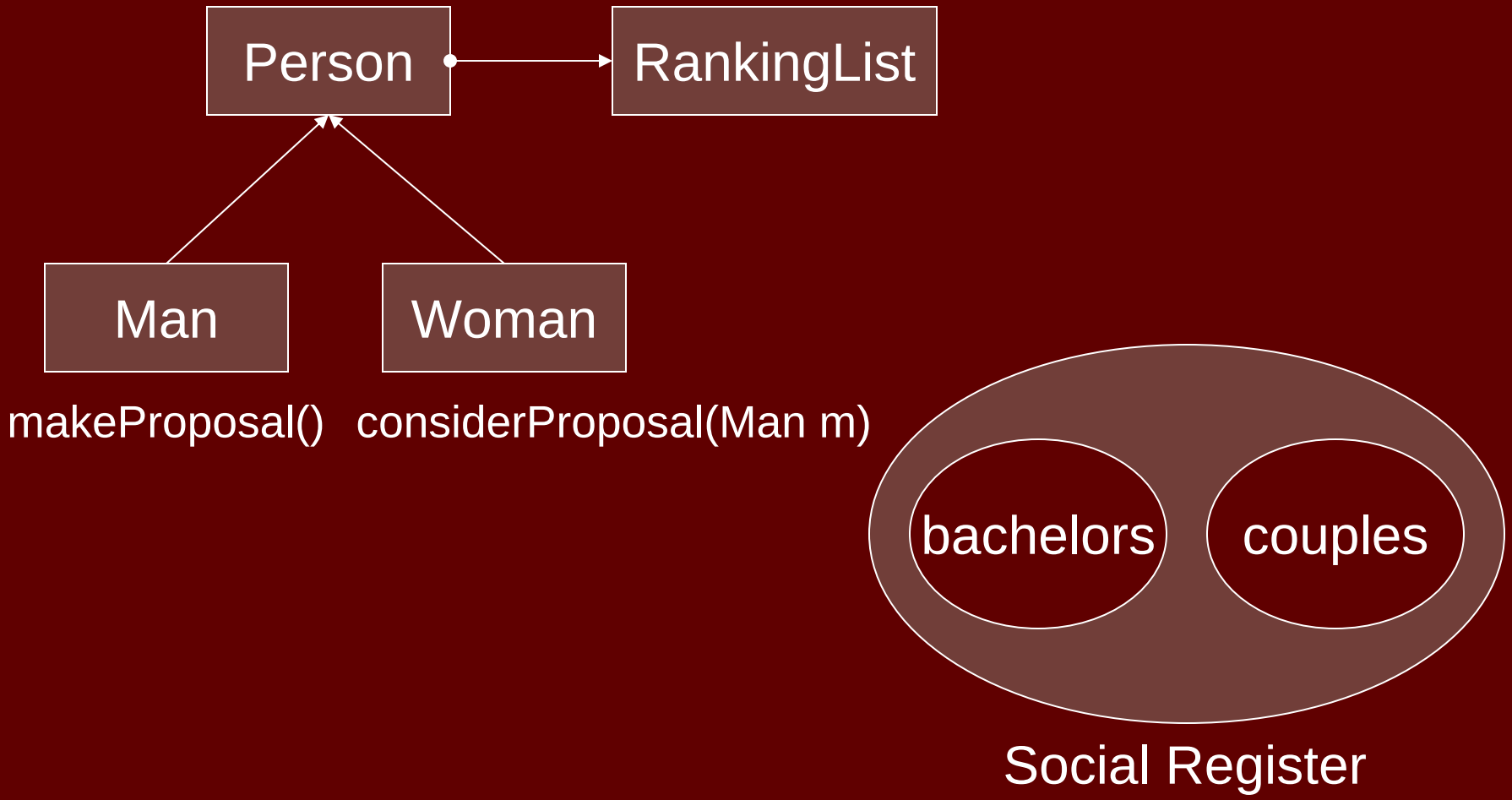
In other words: An arrangement is *stable* if there aren't two people who would leave their spouses and run off together.

* *Stability* is a property of the *set of pairs*, not just one pair.

Ideas?

General Strategy

High Level Design



SM Algorithm

Social Register: main method

While (the set of bachelors is not empty)

Let a bachelor make a proposal

Man: makeProposal() method

Remove W, the top woman, from my rankingList

Ask W to consider a proposal from me

Woman: considerProposal(Man *m*) method

If *m* is on my rankingList

Remove *m and all below m* from my rankingList

Ask the social register to engage me to *m*, breaking any previous engagement if necessary.

Correctness = Safety + Liveness

Safety:

The algorithm won't give a wrong answer.

Liveness:

The algorithm eventually gives an answer.

How do we argue that the SM Algorithm is correct?

SM Safety

Proof by contradiction:

1. Suppose the arrangement produced is not stable.
2. There exist M and W that prefer each other over their own spouses.
3. M must have proposed to W at some point.
4. Either W accepted or she didn't:

   (4a) If W accepted, she would not have dumped him later for a man she ranked below M. So, she is engaged to M, a contradiction.

   (4b) If W didn't accept, she was already engaged to someone she ranked higher than M. So she doesn't really prefer M, a contradiction.
5. Both cases lead to a contradiction; the supposition is false.

SM Liveness

1. The algorithm terminates if there are no bachelors.
2. The number of men and women are equal.
3. The number of bachelors never increases.
4. Whenever a woman becomes engaged for the first time, the number of bachelors decreases.
5. At each step in the algorithm, some man's ranking list decreases in length. So, if the algorithm doesn't terminate, eventually at least one man's list becomes empty.
6. If a man's ranking list becomes empty, he will have proposed to every woman. But an unengaged woman always accepts, so all women are engaged.
7. By (1), (2) and (6), the algorithm terminates.

Does SM favor men or women?

Consider what happens if…

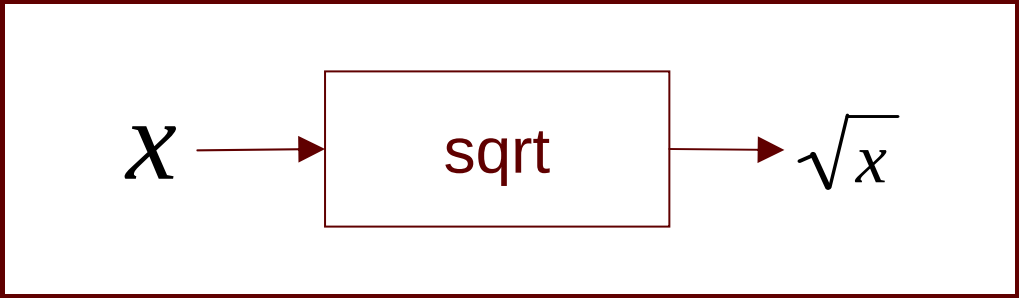
all men have a different first choice.

Consider what happens if…

all women have a different first choice.

A variation of this algorithm is used to match medical school graduates to hospitals for residency. The system is set up by the hospitals. Do you think the hospitals play the role of the men or the women?

The Root Finding Problem



Rewriting the problem:

$$Find: \quad y \cong \sqrt{x}$$

$$\Rightarrow y \times y \cong x$$

$$\Rightarrow y \cong x / y$$

So… given *x*, we want to find *y* such that *y* is close to *x/y*.

sqrt: Given *x*, guess *y* such that *y* is close enough to *x*/*y*

```
double guess(double x, double y) {
    if (closeEnough(y, x/y))
        return y;
    else
        return betterGuess(x, y);
}

double sqrt(double x) {  return guess(x, 1);  }
```

Better Guess?

Q: How to bring y and x/y closer together?

A: Let y be the average of y and x/y.

```
double betterGuess(double x, double y) {
   return (y + x/y) / 2;
}
```

Close Enough?

Q: What is close enough?

```
boolean closeEnough(double a, double b) {
   return abs(a-b) < 0.01;
}
```

a	b	closeEnough?
9	8.993	true
1,000,000,472	1,000,000,471	false
0.00001	0.009	true

Close Enough?

Q: Other ideas?

A: How about a *ratio* close to 1?

```
boolean closeEnough(double a, double b) {
   return abs(1- a/b) < 0.01;
}
```

a	b	closeEnough?
9	8.993	true
1,000,000,472	1,000,000,471	true
0.00001	0.009	false

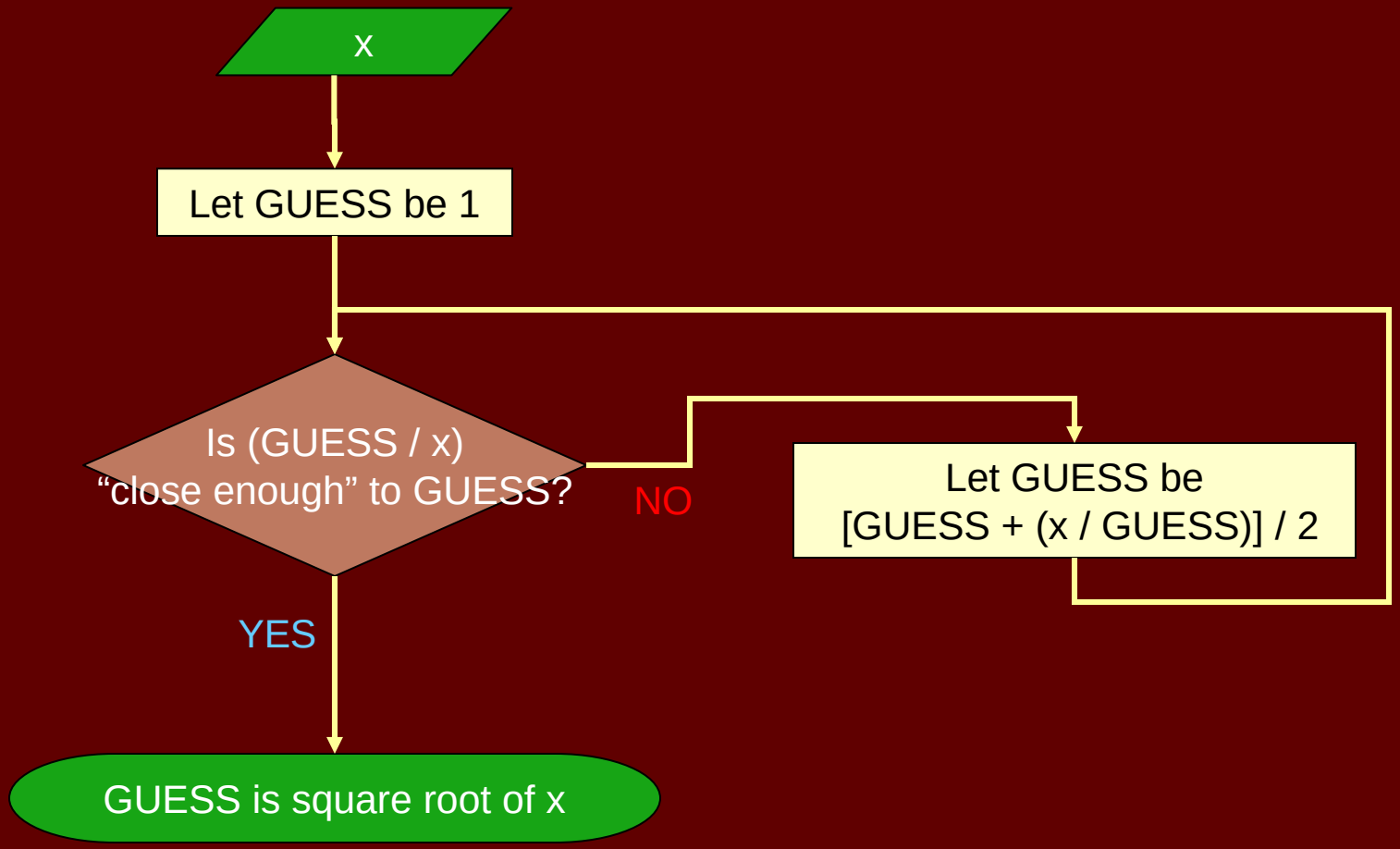
Putting it all together: often harmful*

```
double sqrt(double x) {  return guess(x, 1);  }

double guess(double x, double y) {
   if (abs(1 - y/(x/y) < 0.01)
        return y;
   else
        return (y + x/y) / 2;
}
```

*Removing the subroutine calls hurts readability.

How to Calculate a Square Root



The Greatest Common Divisor

the largest int *k* that divides *m* and *n* with no remainder

Compare Three Algorithms

1. Brute Force
2. Euclid's GCD Algorithm
3. Dijkstra's GCD Algorithm

For simplicity, assume m > n > 0.

Brute Force GCD Algorithm

Idea: Starting with n, keep trying smaller values until one works.

```
int gcd(int m, int n) {
   int k = n;
   while (m%k  !=  0 || n%k != 0)
        k = k - 1;
   return k;
}
```

gcd(21,14)

14
13
12
11
10
9
8
7

Euclid's GCD Algorithm

Idea:

$$for\ m > n > 0, GCD(m,n) = \begin{cases} n & if\ \ m\%n = 0 \\ \gcd(n, m\%n) & otherwise \end{cases}$$

Why does common divisor also divide m%n? First, rewrite *m*…

$$m = n * \lfloor m/n \rfloor + m\%n$$

Now, let *d* be any integer that divides both *m* and *n.*

$$\frac{m}{d} = \frac{n * \lfloor m/n \rfloor}{d} + \frac{m\%n}{d}$$

integer integer integer also!

Euclid's GCD Algorithm

$$for\ m > n > 0, GCD(m,n) = \begin{cases} n & if\ \ m\%n = 0 \\ \gcd(n, m\%n) & otherwise \end{cases}$$

```
int gcd(int m, int n) {
    if ((m % n) == 0)
        return n;
    else
        return gcd(n, m % n);
}
```

gcd(21,14) --> gcd(14,7) --> 7

gcd(468,24) --> gcd(24,12) --> 12

gcd(135,19) --> gcd(19,2) --> gcd(2,1) --> 1

Dijkstra's GCD Algorithm

Idea: If *m*>*n*,
GCD(*m*,*n*)
== GCD(*m*-*n*,*n*).

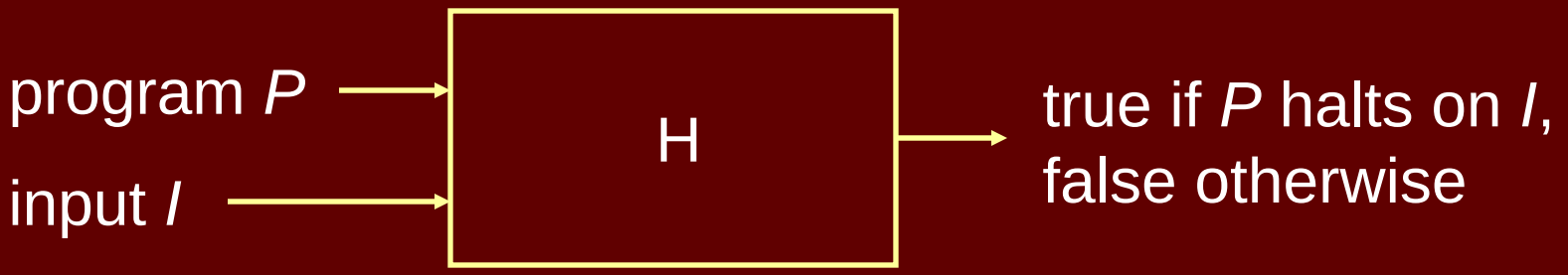
$$for\ m,n>0, \gcd(m,n)=\begin{cases} m & if\ \ m=n \\ \gcd(m-n,n) & if\ \ m>n \\ \gcd(m,n-m) & if\ \ m<n \end{cases}$$

```
int gcd(int m, int n) {
    if (m == n) return n;
    else if (m > n) return gcd(m-n, n);
    else return gcd(m, n-m);
}
```

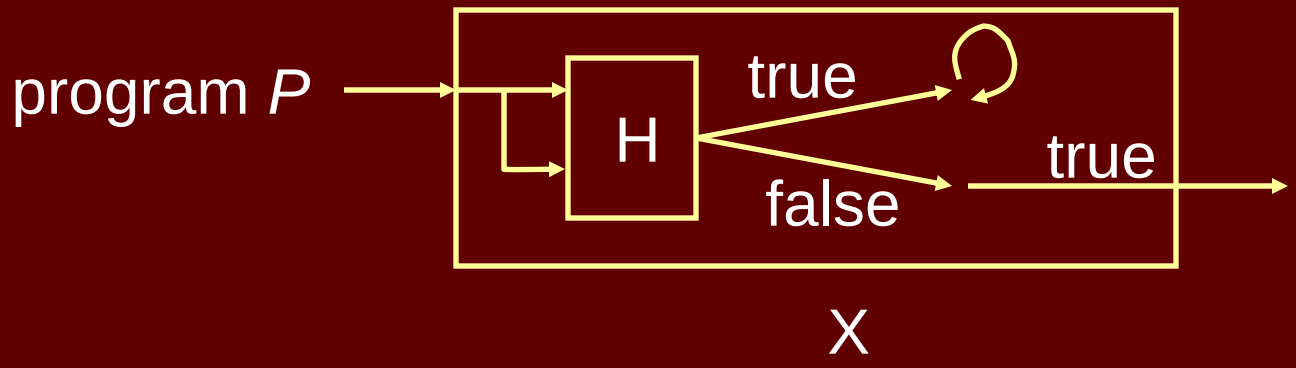
gcd(21,14) --> gcd(14,7) --> 7

gcd(468,24) --> gcd(444,24) --> gcd(420,24) --> … ->
gcd(36,24) --> gcd(12,24) --> gcd(12,12) --> 12

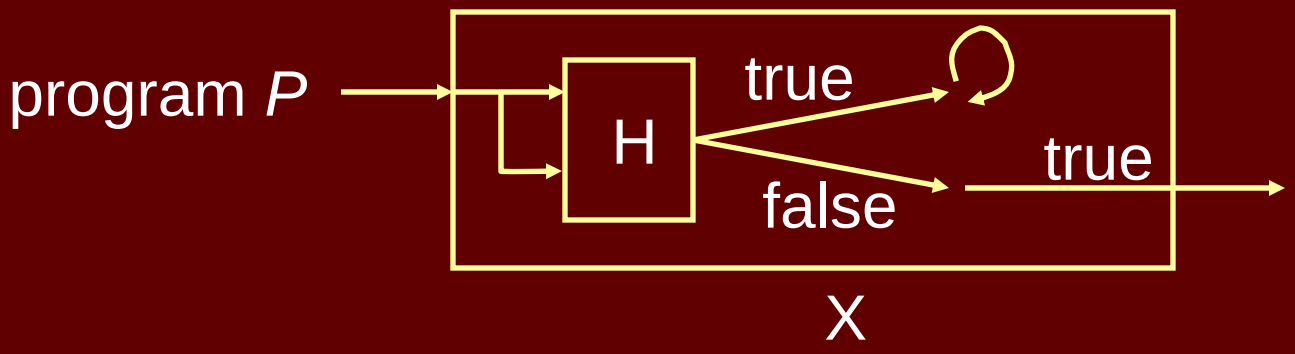
The Halting Problem



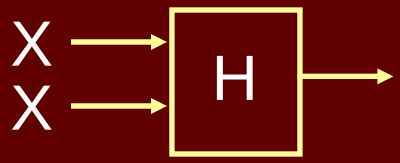
Suppose we could build program H.
Then we could build program X:



The Halting Problem (cont.)



Now, suppose we try:



If H says "true," then X halts given itself as input, but then H would answer true within X and X would not halt!

If H says "false." then X doesn't halt given itself as input, but then H would answer "false" within X and so X would halt!

Therefore, it is impossible ***to solve the halting problem!***