Project: Embedded SMC

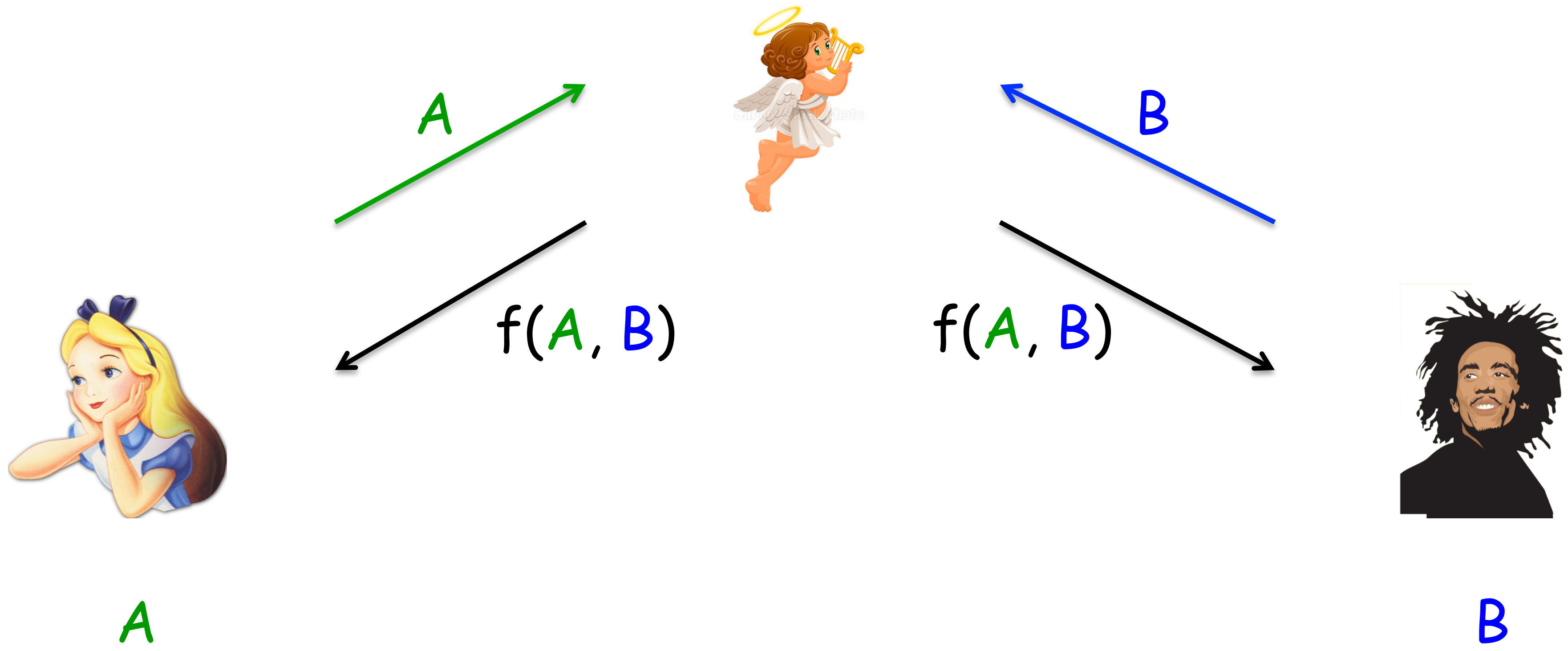
What is Secure Computation [SMC]

A B

Compute $f(A, B)$

Without revealing A to Bob and B to Alice

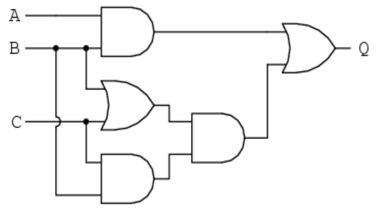
Using a Trusted Third Party



Compute f(A, B)

Without revealing A to Bob and B to Alice

Secure Computation Eliminates Trusted Third Party



Cryptographic Protocol

A

B

Compute f(A, B)

Without revealing A to Bob and B to Alice

Secure Computation Examples

- Richest Millionaire
  - Without revealing salaries
- Nearest Neighbor
  - Without revealing locations
- Auction
  - Without revealing bids
- Private Set Intersection
  - Without revealing sets

Millionaires

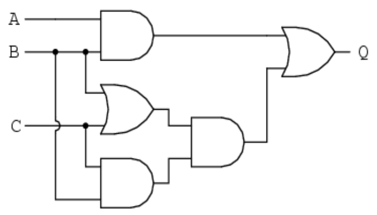
```
bool bob_is_richer(int bob_net_worth, int alice_net_worth) {
  if (bob_net_worth > alice_net_worth) {
    return true;
  } else {
    return false;
  }
}
```

Poor abstraction

- Boolean circuits are not a good mode of programming.
- Millionaires circuit description:

```
n 2
d 532 26 434
i 0 2 13
i 1 14 25
o 0 220 231
o 1 232 243
v 0 1
v 1 1
g 0 0 -1 -1 222 243 242 241 240 239 238 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220 219 218 217 216 215
214 213 212 211 210 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191 190 189 188 187 186 185 184 183 182
181 180 179 178 177 176 175 174 173 172 171 389 388 170 169 168 167 166 165 164 163 162 161 160 159 158 157 156 155 154 153 152 151
150 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133 132 131 130 129 128 127 126 125 124 123 122 121 120 119 118
117 116 115 114 113 112 111 110 245 244 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82
81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35
34 33 32 31 30 29 28 27 26
g 1 0 -1 -1 53 436 440 444 448 452 456 460 464 468 472 476 480 484 488 492 496 500 504 508 512 516 520 524 528 160 292 296 300 304 308
312 316 320 324 328 332 336 340 344 348 352 356 360 364 368 372 376 380 384 64 63 62 50
g 2 0 -1 -1 1 26
g 3 0 -1 -1 1 27
g 4 0 -1 -1 1 28
g 5 0 -1 -1 1 29
g 6 0 -1 -1 1 30
g 7 0 -1 -1 1 31
```

… several hundred more lines (gates)



Boolean Circuit

Poor abstraction

- Need to mix secure computation and local computation (for user input, output to terminal, graphics, etc).

One Solution: design a new language

- See Wysteria.
- Problem: new languages are feature-poor and have little supporting libraries (if any)
  - Especially for user interaction, graphics, etc.

Project: SMC Embedding

- Embed SMC in an existing, established language.
  - Usability: (as much as possible) write secure computations in the same style and syntax as the host language

```
int my_worth;
scanf(“%d”, &my_worth);
bool bob_richer =        **MAGIC** bob_is_richer(?, my_worth) **MAGIC**

if (bob_richer) {
        printf(“bob is richer”);
} else {
        printf(“I am richer”);
}
```

Project: SMC Embedding

- Options:
  - Monads in haskell?
  - *Well designed* class interface in Java?
  - Meta-programming?
  - **Something you are good with?**
- No crypto required, unless you really want to

Two Projects on probabilistic programming

Probabilistic programming

- Higher-level view of programs with randomness (or random input).

```
bool is_rand_even() {
  if (rand() % 2 == 0) {
    return true;
  } else {
    return false;
  }
}
```

- Typical view: samples
  - true, true, false, false, true, true, true …

Probabilistic programming

- Higher-level view of programs with randomness (or random input).

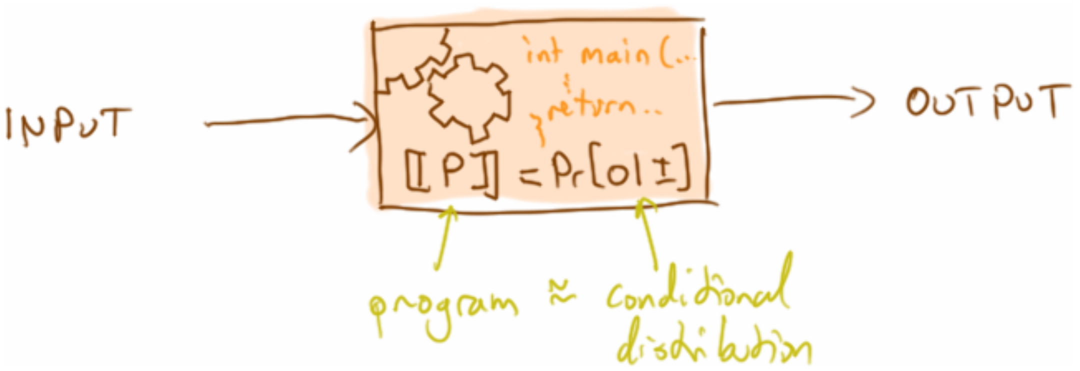
```
bool is_rand_even() {
  if (rand() % 2 == 0) {
    return true;
  } else {
    return false;
  }
}
```

- Probabilistic view: probability distribution
  - Pr[R = true] = Pr[R = false] = 0.5

Probabilistic programming

- A program is a conditional distribution.

```
bool is_even(int i) {
  if (i % 2 == 0) {
    return true;
  } else {
    return false;
  }
}
```


INPUT
int main(...
{
} return..
$[\![P]\!] = \Pr[O \mid I]$
OUTPUT
program ≈ conditional distribution

Inference

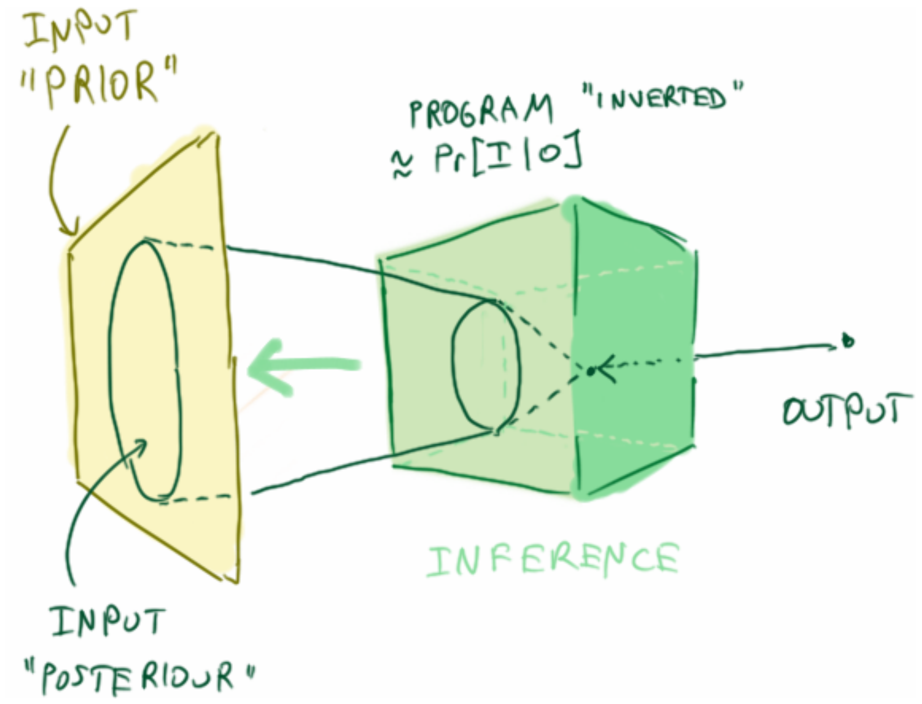
- A program is a conditional distribution.

```
int x = rand() % 1000;

bool is_even(int i) {
  if (i % 2 == 0) {
    return true;
  } else {
    return false;
  }
}

observe (is_even(x) = true);
```

Pr[X=0] = Pr[X=2] … = 1/500
Pr[X=1] = Pr[X=3] … = 0



Inference

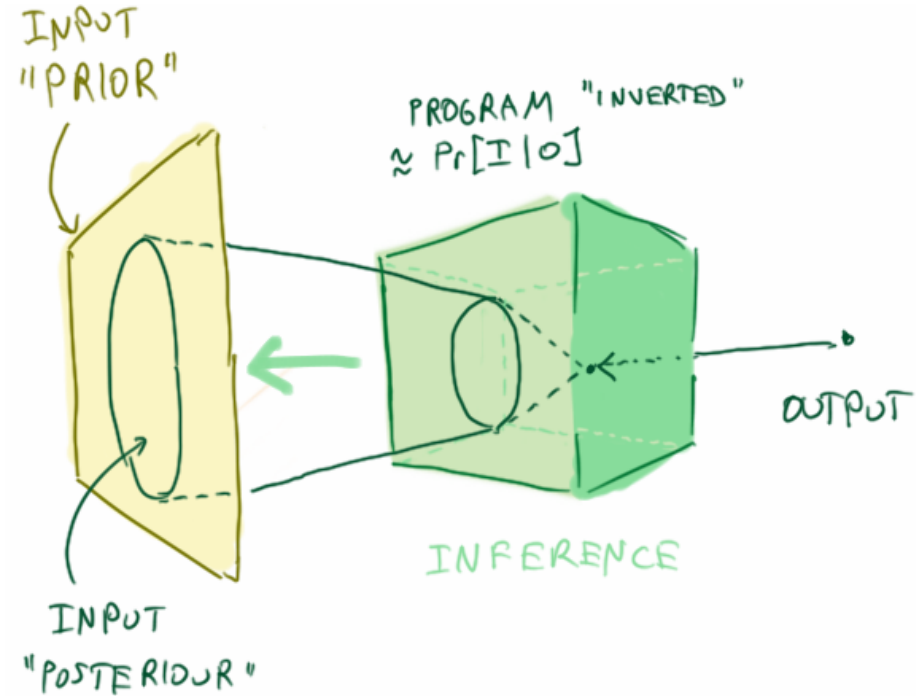
- Automate non-obvious probabilistic inference.

```
int x = (rand() % 1000) + (rand() % 1000);

bool is_even(int i) {
  if (i % 2 == 0) {
    return true;
  } else {
    return false;
  }
}

observe (is_even(x) = true);

Pr[X] = ???
```



Uses

- Machine learning
  - Write down model as a program with missing parameters, infer them based on observation.
- Verification
  - What is the probability that this program has this inconvenient value?
- Security
  - What does the adversary know about a X given they learn output of C(X)?

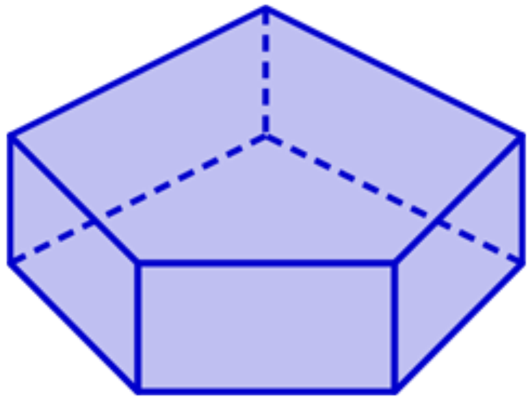
Difficult

- Naïve implementation is no good:
  - (large state spaces): Pr[X=0] = Pr[X=2] … = 1/500
- Solution: use “clever” representations of distributions.
  - Graphical models
  - Decision diagrams
  - Other terms we know nothing about
  - Abstract domains

Abstraction

- Small description of a large state space:

```
int x = rand() % 1000;
```



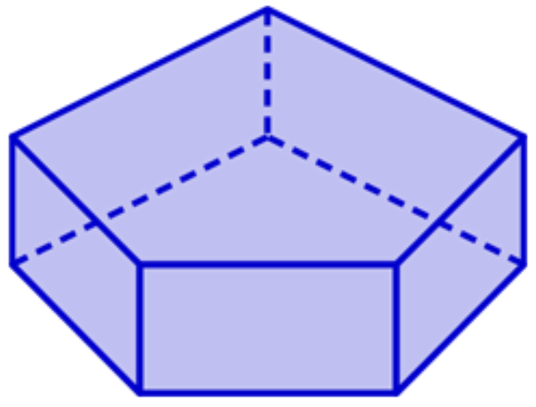
- Naïve:
  - Pr[X=0] = 1/1000, Pr[X=1] = 1/1000 …
- Abstract:
  - Pr[0 <= X < 1000] = 1/1000
- Abstract interpretation:
  - Evaluate program on an abstract set of states (instead of a single one).

Project 1: probabilistic abstract interpretation for functional language

- Existing abstract interpreter:
  - Imperative language, no types beyond int, not data type constructions
- Formalize the abstract interpretation for a functional equivalent with:
  - More types: strings, etc.
  - Data type constructors.
- Implement it?

Abstraction

- Current implementation is based on convex polyhedra (like intervals)
- Good for: int x = rand() % 1000; (0 <= X < 1000)
- Bad for: (observe is_even(x) = true)



Abstraction

- Satisfyability modulo theories [SMT]
  - Represent program states in terms of logical formulas as predicates from some “theory” (like linear inequalities, integers modulo X, etc.).

(0 <= x < 1000) and (x = 0 mod 2)

Theory of integer linear arithmetic

Logical connectives

Project 2: SMT-based abstraction

- Use SMT formulas in representation of probability distributions instead of convex polyhedra.
- Formalize the probabilistic abstract interpretation based on SMT formulas.
- Implement it?