



Pluggable Domains for C Dataflow Analysis

Nathan Coopriider and John Regehr

{coop, regehr}@cs.utah.edu

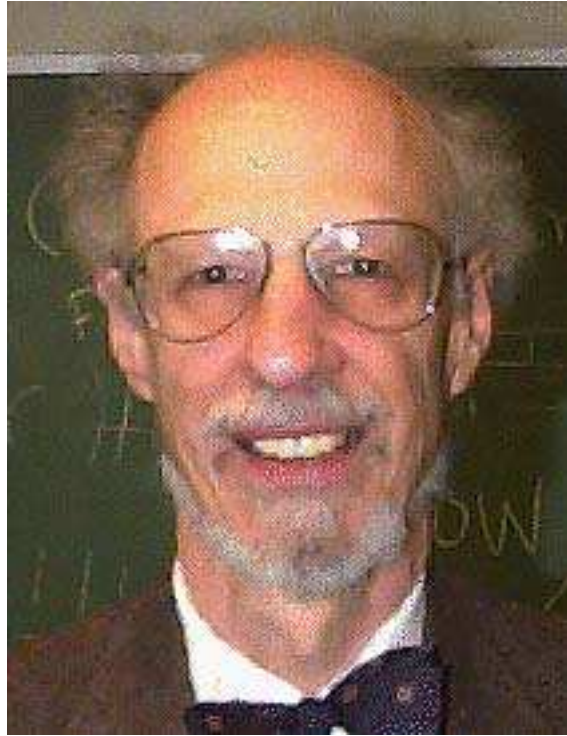
School of Computing, University of Utah



What it is all about

- We read texts on static analysis, program analysis, and abstract interpretation
- Covered lattices, fixpoints, partial evaluation, symbolic execution, transfer functions, CFGs
- \top (no possible values), \perp (all possible values)
- So what?
- This paper is about cXprop, our implementation of these ideas with an eye towards pluggable abstract domains

Shameless plug:



I will give a presentation about

Fernando J. Corbató

and the talk he gave when receiving his Turing award:

On Building Systems That Will Fail

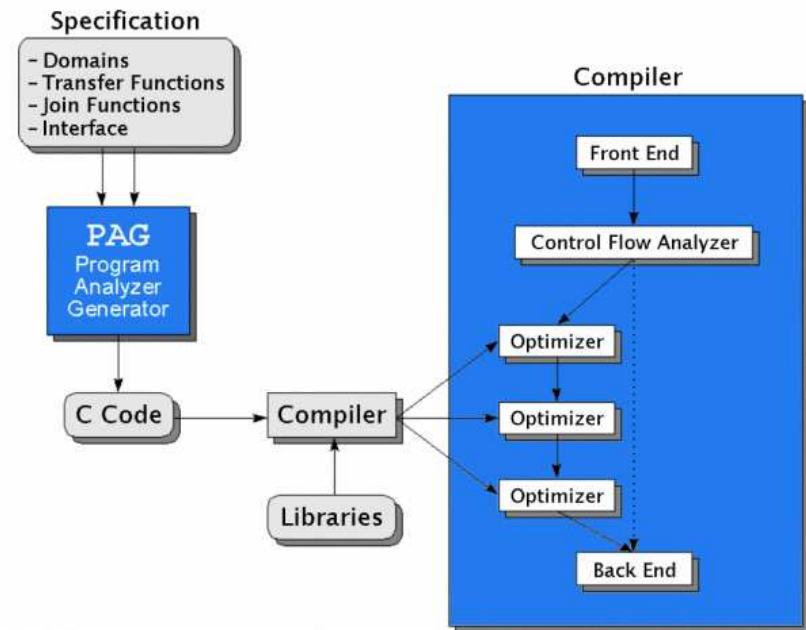
3:05 pm tomorrow (Friday) here (LCR)

A few other static analysis tools:

- Program Analyzer Generator (PAG)

- Needs from the user:


- lattice
 - transfer functions
 - language description
 - fixpoint solution method

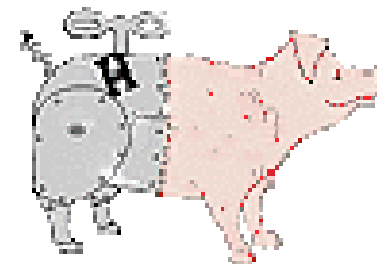


- McCAT


- Generalized Constant Propagation (GCP)

- Machine SUIF

- based on 
 - different interface



CIL

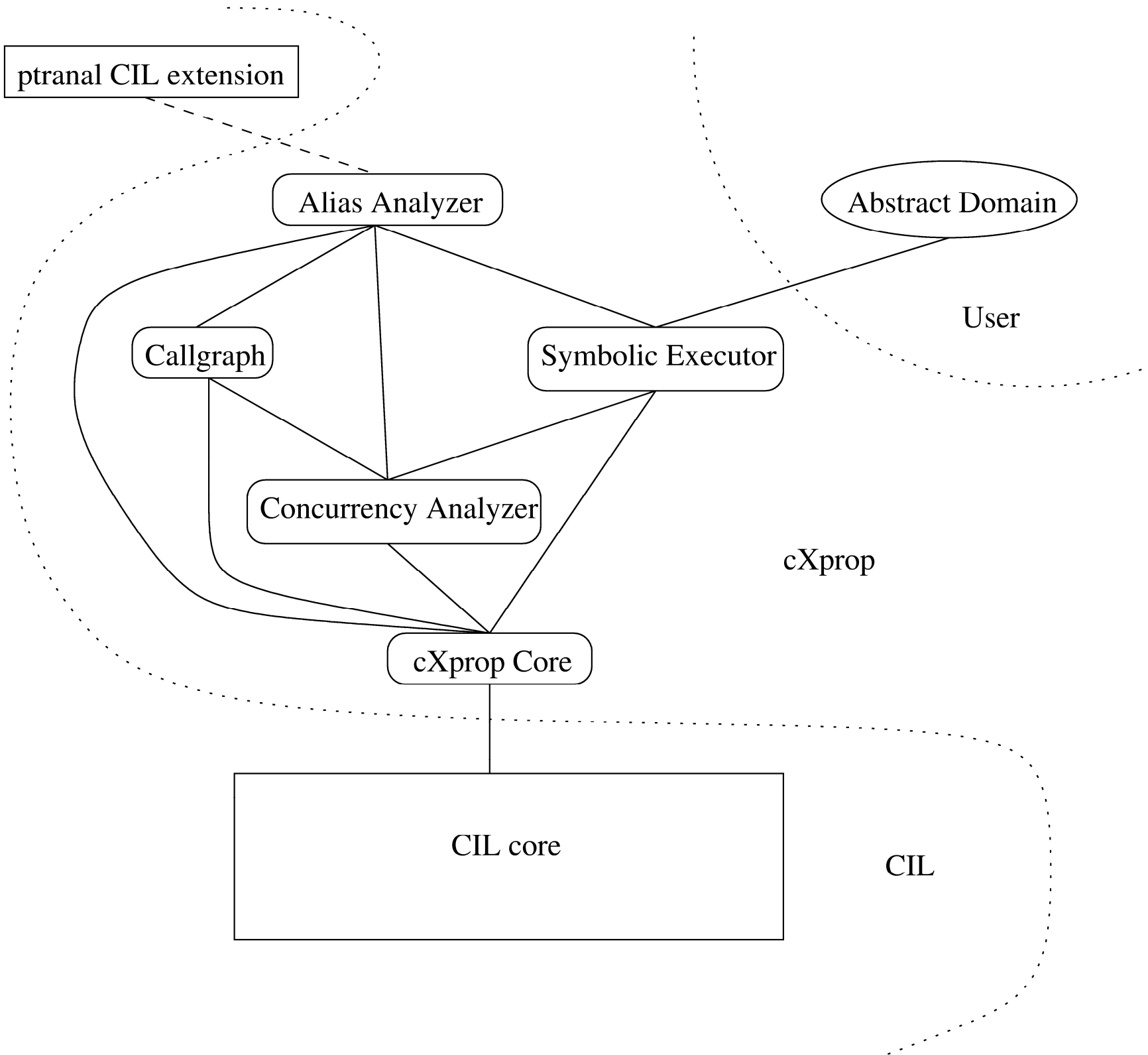
- **C** Intermediate Language – developed at UCB
- Cleans up **C** to a few core constructs
 - removes syntactic sugar (like “->” notation)
 - arrays become pointers
 - all loops become while loops
- Works on real programs
 - handles ANSI-**C** *Microsoft* **C**, and GNU **C**
 - SPEC 95, linux kernel,  , bzip

C : A million and one ways . . .

- To shoot yourself (and our analysis) in the foot
 - Stack manipulations
 - External calls
 - Floating point
 - Order of evaluation
 - Concurrency
 - Missing returns



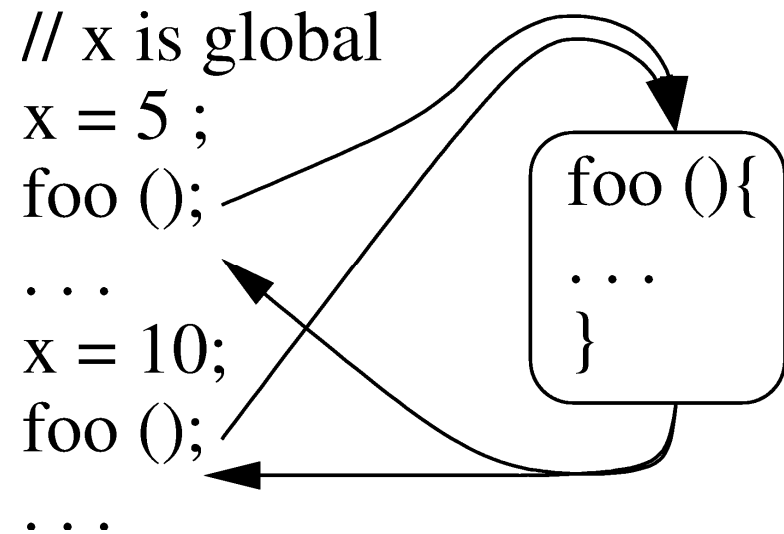
**c
X
p
r
o
p**



Two important pieces:

- Write sets

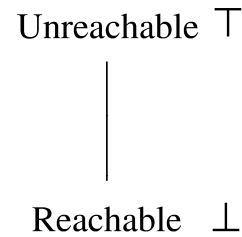
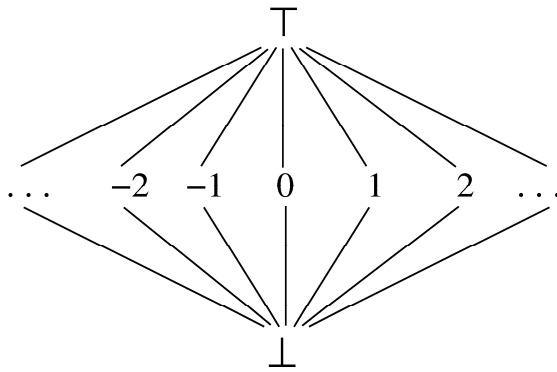
- in figure, x is \perp inside `foo`
- but what if `foo` does not use x ?



- Pointer analysis

- what to do with reading or writing pointers
- function pointer calls

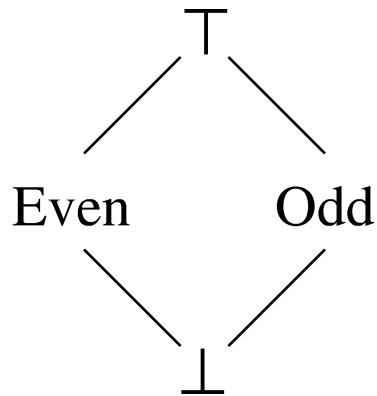
Constant



```
let mult d1 d2 tp =  
  match d1, d2 with  
  | Constant(z), _  
  | _, Constant(z) when (isZero z) ->  
    Constant (zero)  
  | Bottom, _  
  | _, Bottom -> Bottom  
  | Constant(e1), Constant(e2) ->  
    conc_to_abs (BinOp(Mult,e1,e2,tp))
```

```
1 int tricky () {  
2   int x = 1;  
3   int count = 0;  
4   do {  
5     int b = x;  
6     if (b != 1)  
7       x = 2;  
8     count += x;  
9   } while (count < 10);  
10  return x;  
11 }
```

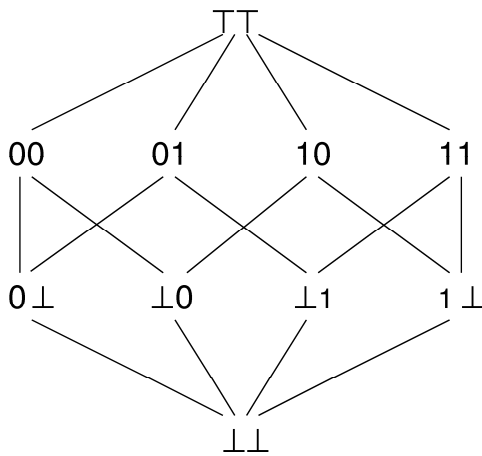
Parity



```
let plusa (d1:t) (d2:t) tp =  
  match d1, d2 with  
  | Bottom, _  
  | _ , Bottom -> Bottom  
  | Even, Even  
  | Odd, Odd -> Even  
  | Even, Odd  
  | Odd, Even -> Odd
```

```
1 int tricky () {  
2   int x = 1;  
3   int count = 0;  
4   do {  
5     int b = x;  
6     if (b != 1)  
7       x = 2;  
8     count += x;  
9   } while (count < 10);  
10  return x;  
11 }
```

Bitwise



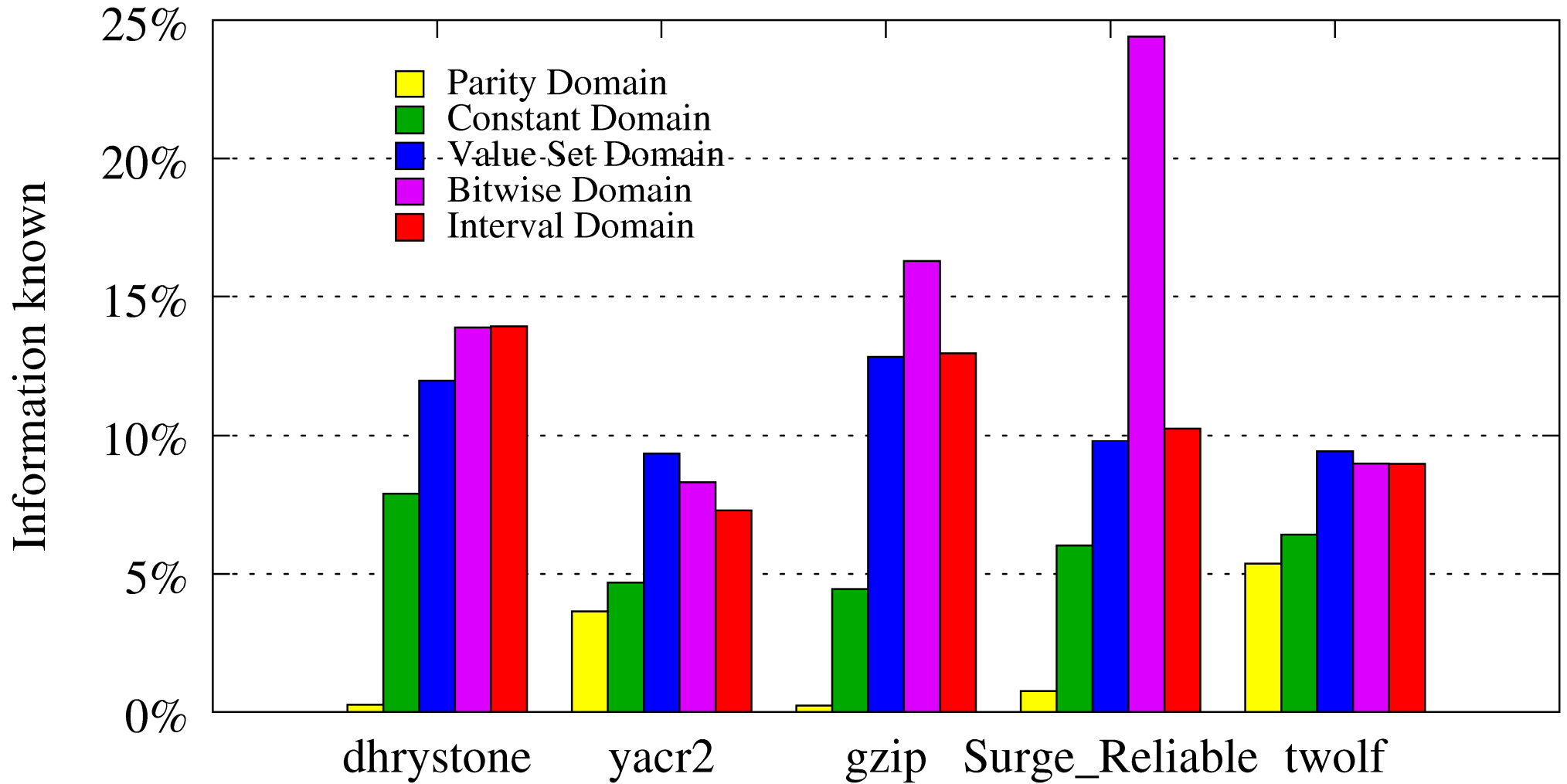
```
let Inot (d, dk) tp =  
  if ((dk = no_bottoms)&&(d = I.zero))  
  then TbTrue  
  else if (I.logand d dk) <> I.zero then TbFalse  
  else TbBottom
```

```
1 int tricky () {  
2   int x = 1;  
3   int count = 0;  
4   do {  
5     int b = x;  
6     if (b != 1)  
7       x = 2;  
8     count += x;  
9   } while (count < 10);  
10  return x;  
11 }
```

Information

- *entropy* in information theory (not physics)
- Information
 - j = # of possible values that can be represented
 - k = # of values that might be represented
- Information bits known
 - $\lceil \log_2 j \rceil$ = # of information bits
 - $\lceil \log_2 k \rceil$ = # of information bits unknown
 - $\lceil \log_2 j \rceil - \lceil \log_2 k \rceil$ = # of information bits known

Domain Comparison



Conclusion

- cXprop performs abstract interpretation
- Convenient interface for new abstract domains
- Five domains already implemented:
 - parity, constant, value set, interval, bitwise
- Not all programs analyze the same way
- Pick the right domain for the program
- May be downloaded at:

<http://www.cs.utah.edu/~coop/research/cxprop>