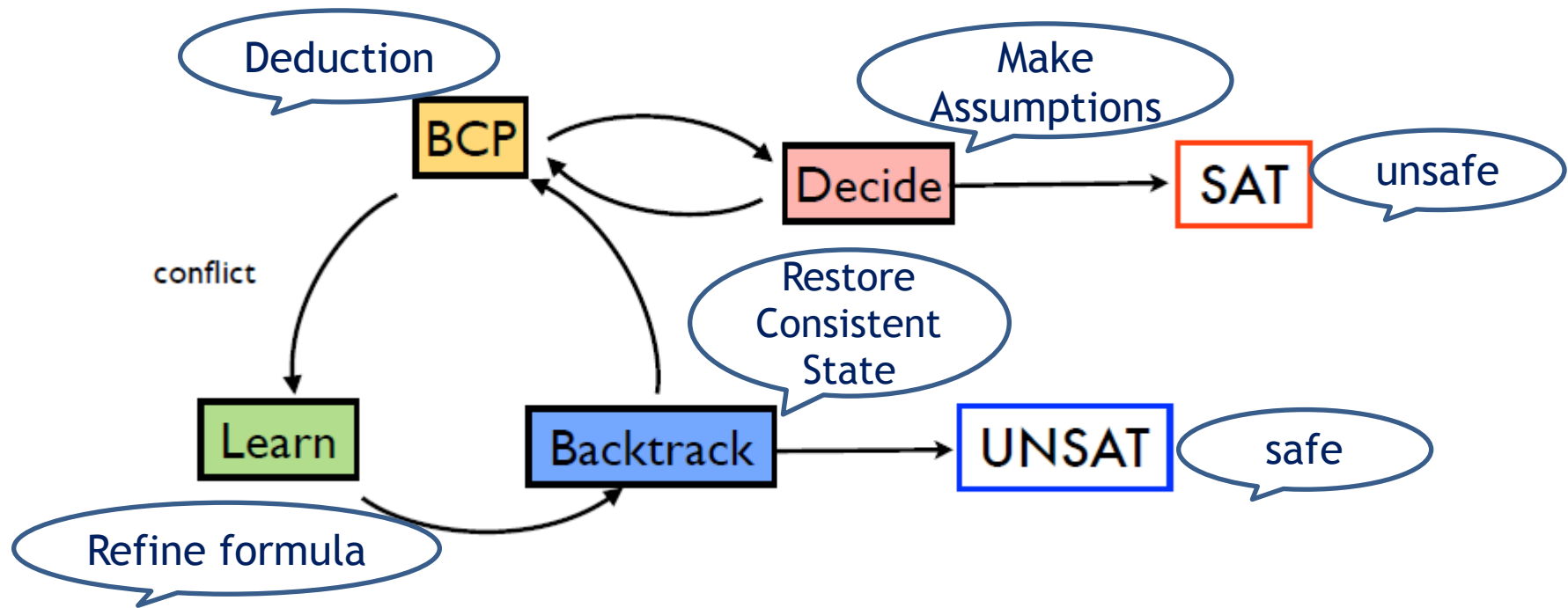


Lifting CDCL to Abstract Lattices

Rajdeep Mukherjee
PhD, University of Oxford



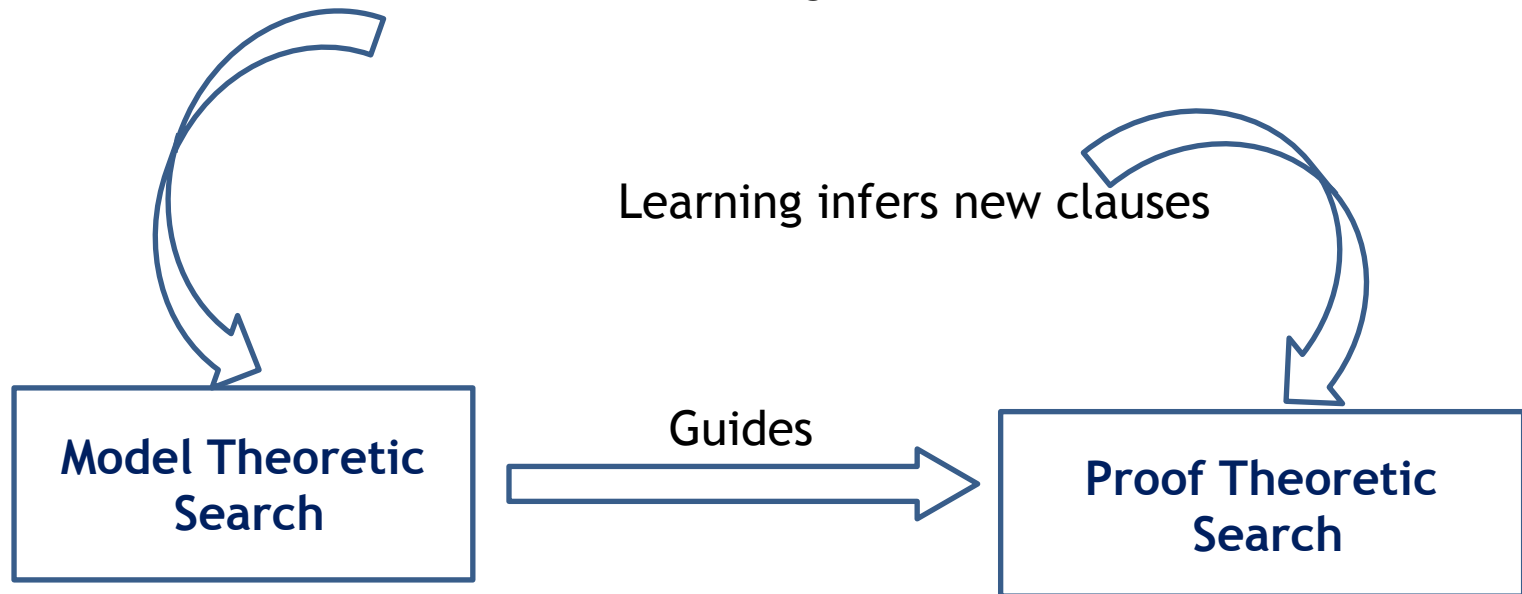
Conflict Driven Clause Learning



**** Partial assignments are main data structure in SAT solvers**

The CDCL Algorithm Summary

BCP + Decision \rightarrow Constructs an assignment



Lifting CDCL to Program Analysis

SAT Solvers:	Precise but inefficient
Abstract Interpreter:	Efficient but Imprecise

How to make SAT solvers more efficient?

→ Choose a domain that's better suited to your problem than the Boolean domain!

How to make Abstract interpretation more precise?

→ Wrap them in the SAT architecture!

Abstract Conflict Driven Clause Learning

Satisfiability Solvers are Abstract Interpreters **

Decision Procedure

Partial Assignments →
Decision →
Unit Rule →
BCP →
Learning →

Abstract Interpretation

Abstract Domain
Restrict range of variables
Best Abstract Clause Transformer
GFP Iteration
Generate program analysis constraint
Implicit form of Trace Partitioning

** Leopold Haller, Abstract Satisfaction, D.Phil, University of Oxford, 2013

** Papers on ACDCL: TACAS 2013, POPL 2013, POPL 2014, FMSD 2014, SAS 2013

About ACDCL

New program analysis that embeds an abstract domain inside Conflict Driven Clause Learning algorithm of modern day SAT solvers.

What is ACDCL?

From AI Point of View → ACDCL is an abstract interpreter that uses Decision and Learning to increase transformer precision

From a decision procedure perspective → ACDCL is a SAT solver for program analysis constraints. It is a strict generalisation of propositional CDCL.

Sketch of ACDCL Algorithm

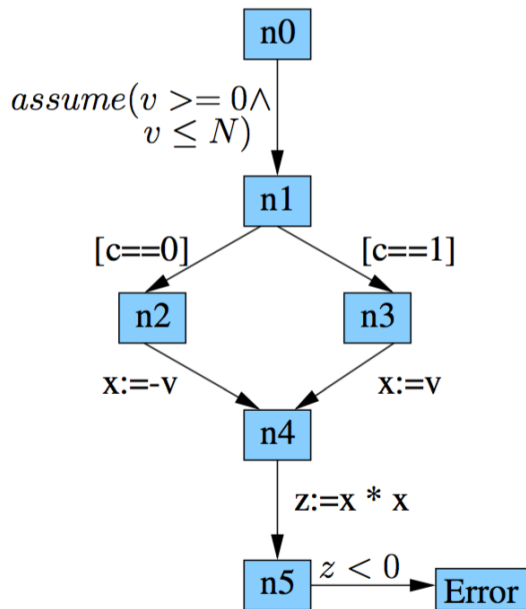
Algorithm 1: Abstract Conflict Driven Clause Learning $ACDCL_{H_P, H_D, H_C}(\mathcal{A})$

input : A program in the form of a set of abstract transformers \mathcal{A} .

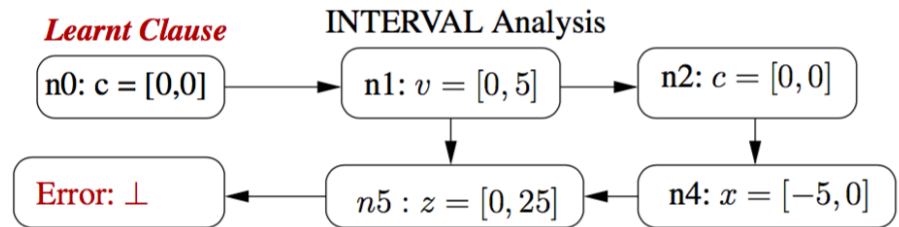
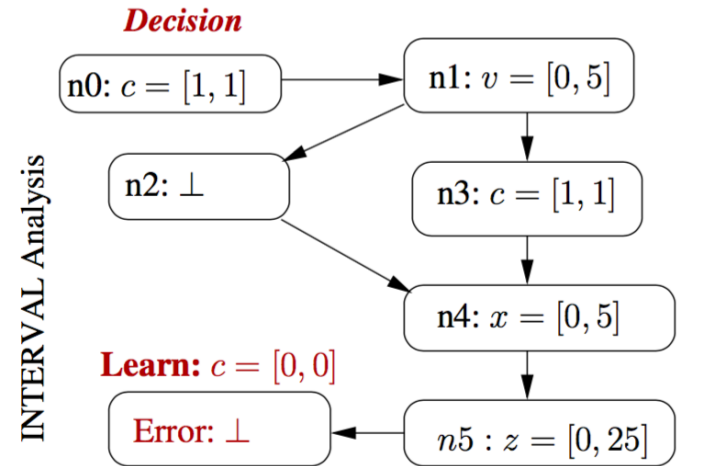
output : The status **safe** or **unsafe**.

```
1  $\mathcal{T} \leftarrow \langle \rangle, \mathcal{R} \leftarrow []$ 
2  $result \leftarrow deduce_{H_P}(\mathcal{A}, \mathcal{T}, \mathcal{R})$ 
3 if  $result = \text{conflict}$  then return safe
4 while true do
5   if  $result = \text{sat}$  then return unsafe
6    $q \leftarrow decide_{H_D}(abs(\mathcal{T}))$ 
7    $\mathcal{T} \leftarrow \mathcal{T} \cdot q$ 
8    $\mathcal{R}[[\mathcal{T}]] \leftarrow \top$ 
9    $result \leftarrow deduce_{H_P}(\mathcal{A}, \mathcal{T}, \mathcal{R})$ 
10  do
11    if  $\neg analyzeConflict_{H_C}(\mathcal{A}, \mathcal{T}, \mathcal{R})$  then return safe
12     $result \leftarrow deduce_{H_P}(\mathcal{A}, \mathcal{T}, \mathcal{R})$ 
13  while  $result = \text{conflict}$ 
14 end
```

Instantiating CDCL with Numerical Abstract Domains

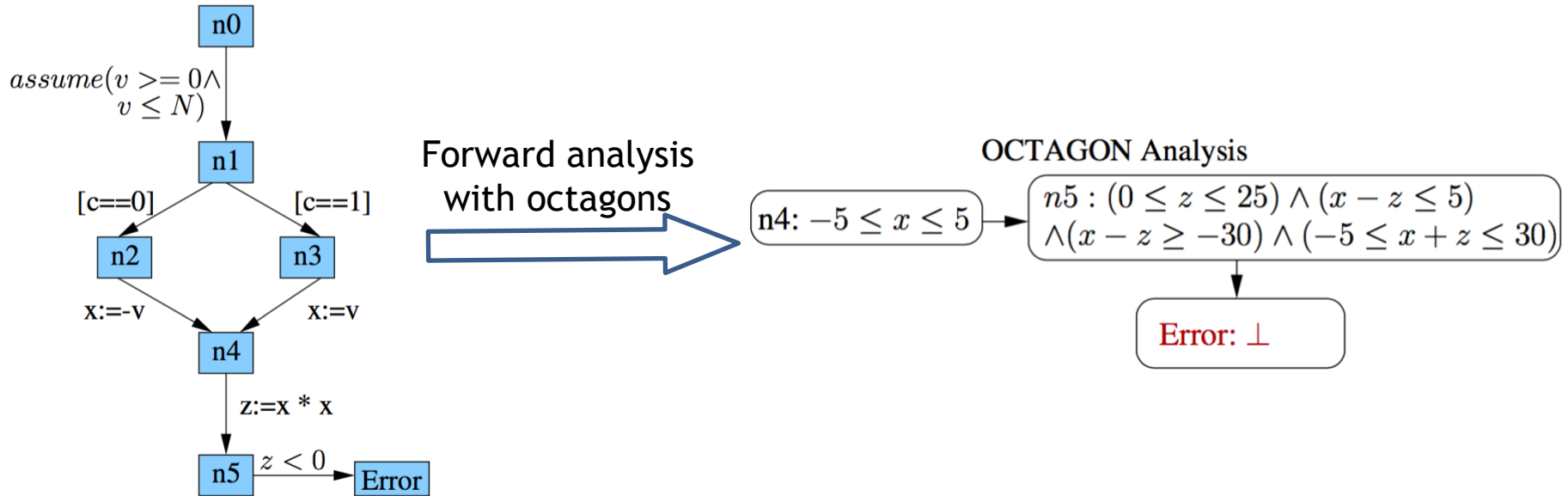


Forward analysis
with intervals



** Astree failed to verify with interval domain, require external hint to recover from imprecision

Instantiating CDCL with Numerical Abstract Domains



For (N=46000)

Solver	decisions	propagations	conflicts	conflict literals	restarts
MiniSat	233	36436	162	2604	2
ACDCL (Interval)	1	17	1	1	0
ACDCL (Octagons)	0	7	0	0	0

SAT solver operates over Boolean Constants Abstract Domain

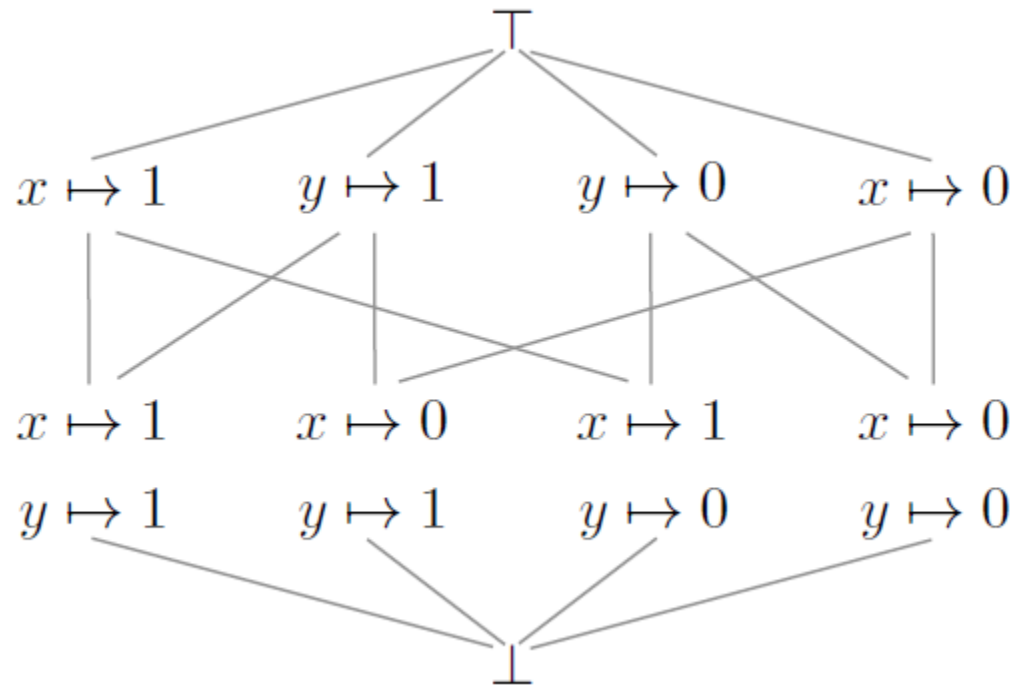
Main data-structure in a SAT solver::
Partial Assignment

Partial Assignment
 $\text{Prop} \rightarrow \{t, f, ?\}$

Three-Valued Logic

Assignments are extended
using deduction and decisions

Boolean Constants Domain



BCP is GFP computation

- ❖ Abstract transformers are sound but imprecise.
- ❖ gfp iteration improves the precision of the abstraction

Theorem (Cousot and Cousot 1979)::

The concretisation of greatest fixed-point computation over abstract post transformer gives more precise deductions compared to the concretisation of abstract post transformer

Example of sAbstract fixed-point

$$\begin{aligned}\varphi &= p \wedge (\neg p \vee q) \\ \text{apost}_{\varphi}(\top) &= \text{apost}_p(\top) \sqcap (\text{apost}_{\neg p}(\top) \sqcup \text{apost}_q(\top)) \\ &= \langle p \mapsto \text{t} \rangle \\ \text{apost}_{\varphi}(\langle p \mapsto \text{t} \rangle) &= \text{apost}_p(\langle p \mapsto \text{t} \rangle) \sqcap (\text{apost}_{\neg p}(\langle p \mapsto \text{t} \rangle) \sqcup \text{apost}_q(\langle p \mapsto \text{t} \rangle)) \\ &= \langle p \mapsto \text{t} \rangle \sqcap (\perp \sqcup \langle p \mapsto \text{t}, q \mapsto \text{t} \rangle) \\ &= \langle p \mapsto \text{t} \rangle \sqcap \langle p \mapsto \text{t}, q \mapsto \text{t} \rangle \\ &= \langle p \mapsto \text{t}, q \mapsto \text{t} \rangle\end{aligned}$$

Properties of Domain elements for lifting CDCL

Property of a Clause Learning SAT solver:

Each non-singleton element of the partial assignment domain can be decomposed into a set of precisely complementable singleton elements

Property of Numerical Abstract Domains:

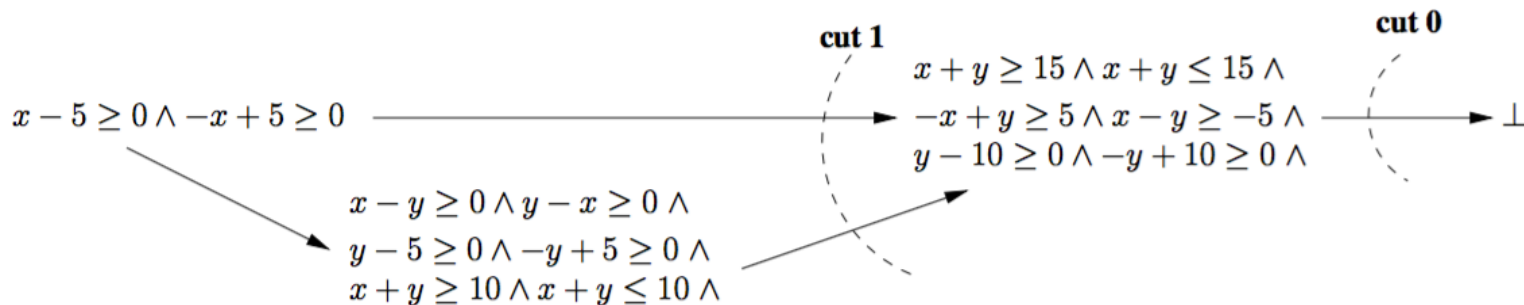
Abstract domains such as Intervals and Octagons are decomposable into half-spaces which have precise complements

$\text{decomp}(2 \leq x \leq 4 \wedge 3 \leq y \leq 5) \rightarrow \{x \geq 2, x \leq 4, y \geq 3, y \leq 5\}$

Abstract Learning

- >> Learning and case based reasoning are different !
- >> A case-based reasoning does proof under each assumption.
- >> Learning generates constraints that preserve error reachability.
- >> ACDCL adds the learnt clause to the set of transformer

Consider a program fragment: $\varphi := (y=x \wedge y=y+x \wedge y \leq 0)$



Use Abstract Conflict Graph Cutting Algorithm to find UIP

DPLL with No Abstract Learning

Input Program

```
if(y <4)
  x = 1;
else
  x = -1;
assert(x!=0);
```

DPLL strategy using forward analysis with Intervals

1. $x : [0, \text{INF}]$
2. $x : [0, \text{INF}], y : [0, \text{INF}]$
3. $x : [0, \text{INF}], y : [5, \text{INF}] \rightarrow \text{PROOF}$
4. $x : [0, \text{INF}], y : [-\text{INF}, 4]$
5. $x : [0, \text{INF}], y : [-\text{INF}, 3] \rightarrow \text{PROOF}$
6. $x : [0, \text{INF}], y : [4, 4] \rightarrow \text{PROOF}$
7. $x : [0, \text{INF}], y : [-\text{INF}, 0] \rightarrow \text{PROOF}$
8. $x : [-\text{INF}, 0]$
9. $x : [-\text{INF}, 0], y : [0, \text{INF}]$
10. $x : [-\text{INF}, 0], y : [5, \text{INF}] \rightarrow \text{PROOF}$
11. $x : [-\text{INF}, 0], y : [-\text{INF}, 4]$
12. $x : [-\text{INF}, 0], y : [-\text{INF}, 3] \rightarrow \text{PROOF}$
13. $x : [-\text{INF}, 0], y : [4, 4] \rightarrow \text{PROOF}$
14. $x : [-\text{INF}, 0], y : [-\text{INF}, 0] \rightarrow \text{PROOF}$

- Bad choice by splitting on 'x' because x is overwritten !
- Made bad splits on 'y' !

CDCL with Abstract Learning

Input Program (P)

```
if(y < 4)
  x = 1;
else
  x = -1;
assert(x != 0);
```



Analysis using ACDL

```
if(y < 4) {
  P;
  assert(x != 0);
}
else {
  P;
  assert(x != 0);
}
```

CDCL style strategy using forward analysis with Intervals

- 1 $x : [0, \text{INF}]$
- 2 $x : [0, \text{INF}], y : [0, \text{INF}]$
- 3 $x : [0, \text{INF}], y : [5, \text{INF}] \rightarrow \text{PROOF}$

Value of x is irrelevant for the proof



Perform
Generalization

1. $x : [0, \text{INF}]$
2. $x : [0, \text{INF}], y : [0, \text{INF}]$
3. $x : [0, \text{INF}], y : [5, \text{INF}] \rightarrow \text{PROOF}$
4. Generalize to $y : [4, \text{INF}] \rightarrow \text{PROOF}$
5. $y : [-\text{INF}, 3] \rightarrow \text{PROOF}$

Proof still works if split
at $y[4, \text{INF}]$ instead of $y[5, \text{INF}]$

Case Splitting:

$y : [-\text{INF}, 3]$ and $y : [4, \text{INF}]$

CDCL learns this is the
only interesting case left

Program and Property Driven Trace Partitioning

Input Program (P)

```
if(y <4)
  x = 1;
else
  x = -1;
assert(x!=0);
```

Analysis using ACDL



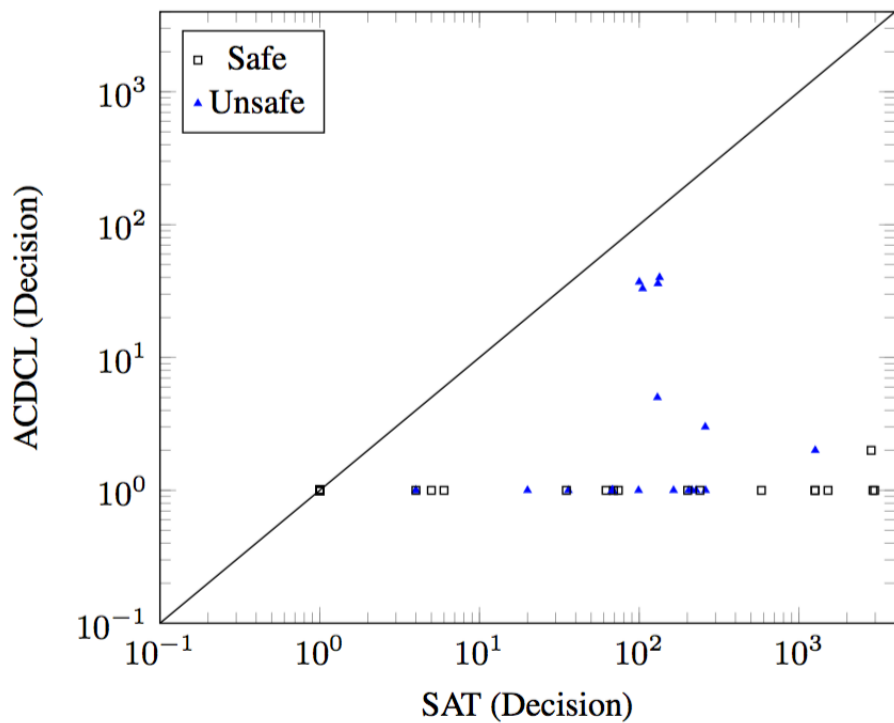
```
if(y <4) {
  P;
  assert(x!=0);
}
else {
  P;
  assert(x!=0);
}
```

CDCL with abstract learning implicitly produces program and property driven trace partitioning.

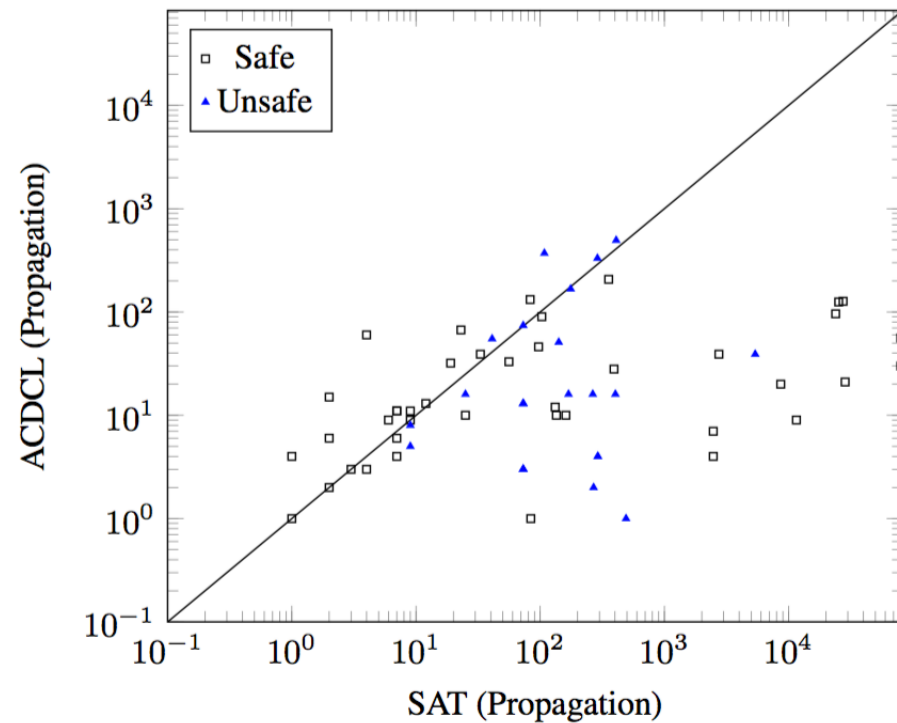
Program Dependent: If the guard were ($y < 10$), ACDL would have generated different partitioning like $y: [-\text{INF}, 9]$, $y: [10, \text{INF}]$

Property Dependent: If the assertion had been ($x < 1$), no splitting would have been needed

SAT versus ACDCL



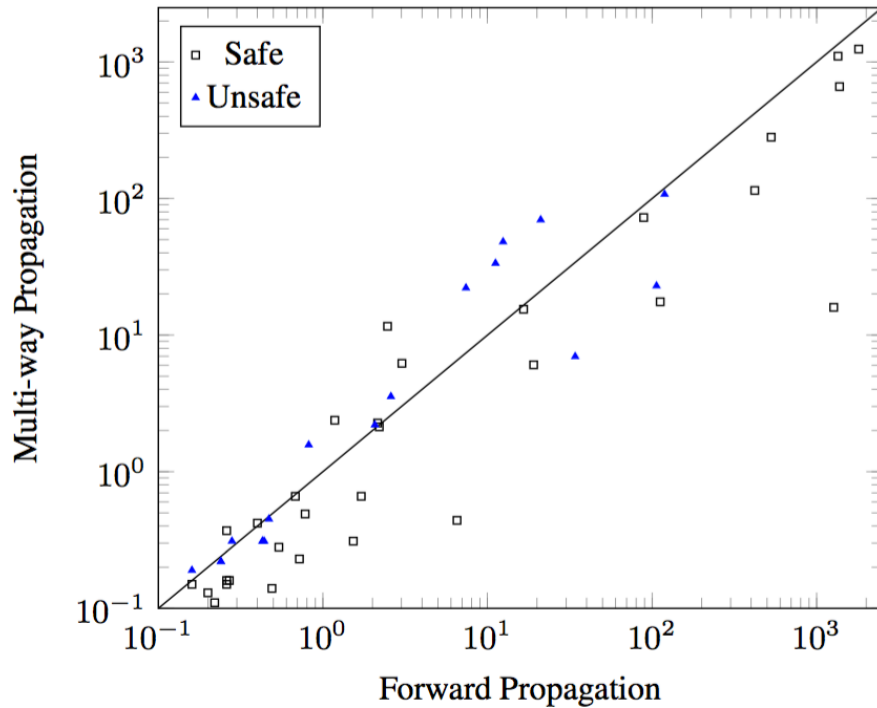
(a)



(b)

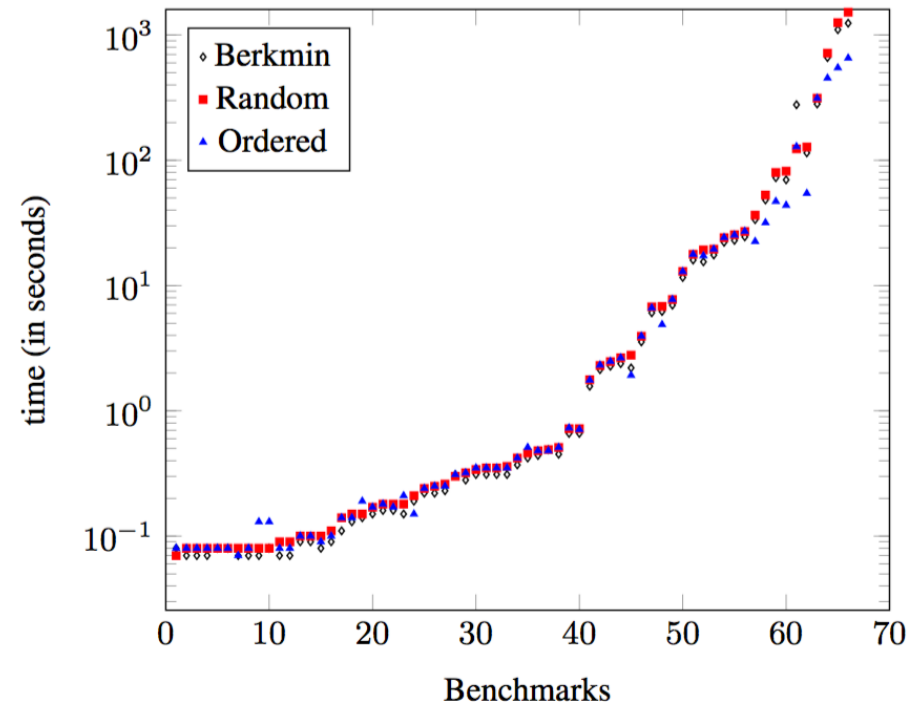
Effect of Decision and Propagation Heuristics

Performance of Propagation Heuristics



(a)

Performance of Decision Heuristics



(b)

Ongoing Work

- >> Efficient Gamma completeness check in Template Polyhedra
- >> Instantiate CDCL over more expressive lattice structures
- >> Generalize Conflict Reasons for Template Polyhedra