Maximum Satisfiability in Software Analysis: Applications & Techniques

Mayur Naik, University of Pennsylvania

Joint work with:

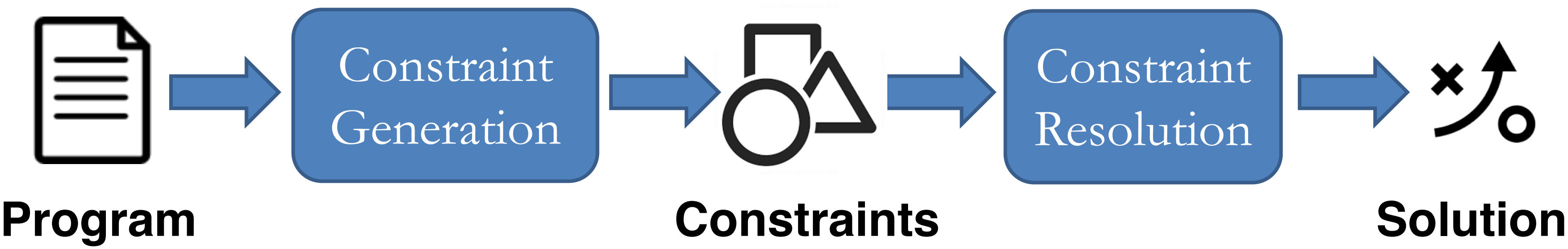
Xujie Si and Xin Zhang
Georgia Tech

Aditya Nori
Microsoft Research

Radu Grigore
University of Kent

Hongseok Yang
University of Oxford

Constraint-Based Software Analysis



- Long history
  - Type constraints, set constraints, SAT/SMT constraints
- Many benefits
  - Separates analysis specification from implementation
  - Allows to leverage sophisticated off-the-shelf solvers
  - Yields natural program specifications
  - …

Challenges in Software Analysis

But constraint-based approach is not well-suited for …

- Balancing trade-offs
  - e.g. precision vs. scalability
- Handling uncertainty
  - e.g. incorrect specifications
- Modeling missing information
  - e.g. incomplete programs

An Emerging Approach

Constraint **Satisfaction** ➡ Constraint **Optimization**

- Balancing trade-offs
  - e.g. precision vs. scalability
- Handling uncertainty
  - e.g. incorrect specifications
- Modeling missing information
  - e.g. incomplete programs

} **Objectives**

The Maximum Satisfiability Problem

SAT:

$$\begin{array}{rcl} a & \wedge & (\mathbf{C1}) \\ \neg a \vee b & \wedge & (\mathbf{C2}) \\ \neg b \vee c & \wedge & (\mathbf{C3}) \\ \neg c \vee d & \wedge & (\mathbf{C4}) \\ \neg d & & (\mathbf{C5}) \end{array}$$

The Maximum Satisfiability Problem

MaxSAT:

	$a \quad \wedge$	(C1)
	$\neg a \vee b \quad \wedge$	(C2)
4:	$\neg b \vee c \quad \wedge$	(C3)
2:	$\neg c \vee d \quad \wedge$	(C4)
7:	$\neg d$	(C5)

=

Subject to	**C1**
	C2
Maximize	**4×C3+2×C4+7×C5**

Solution: a = true, b = true, c = true, d = false
(Objective = 11)

The Maximum Satisfiability Problem

+ Expressive
for our problems

Hard — **Soundness Conditions**

+

Soft — **Objectives**

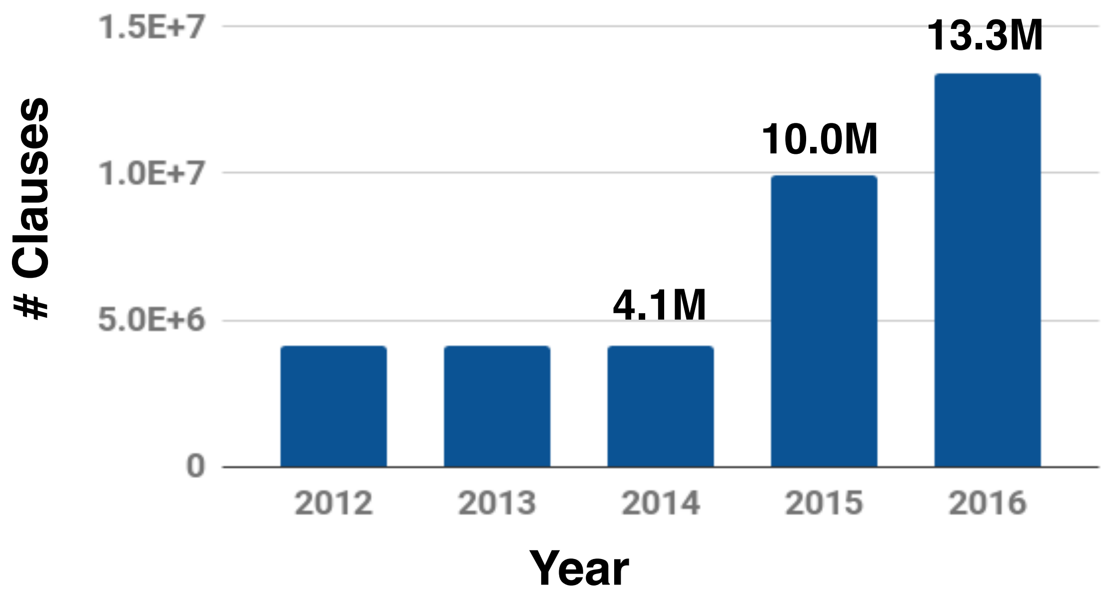
- **Balancing tradeoffs (e.g., precision vs. scalability)**
- **Handling uncertainty (e.g., incorrect specs)**
- **Modeling missing data (e.g., partial programs)**
- …

The Maximum Satisfiability Problem

+ Expressive
for our problems

+ Efficient
(and improving) solvers

Largest instance solved in MaxSAT competition:



The Maximum Satisfiability Problem

+ Expressive
for our problems

+ Efficient
(and improving) solvers

$$
\begin{aligned}
& && \forall x.\ \mathrm{path}(x, x) \\
\wedge & && \forall x, y, z.\ \mathrm{path}(x, y) \wedge \mathrm{edge}(y, z) \Longrightarrow \mathrm{path}(x, z) \\
\wedge & \quad \mathbf{1.5:} && \forall x, y.\ \neg\, \mathrm{path}(x, y)
\end{aligned}
$$

− Cannot concisely specify our problems (lacks quantifiers)

− Loses high-level structure that solvers could exploit

The Maximum Satisfiability Problem

+ Expressive

+ Efficient

How to overcome limitations of MaxSAT while retaining its benefits?

$\forall$ x. path(x, x)

$\wedge$ $\forall$ x, y, z. path(x, y) $\wedge$ edge(y, z) $\Longrightarrow$ path(x, z)

$\wedge$ **1.5:** $\forall$ x, y. $\neg$ path(x, y)

A solution: Markov Logic Network (MLN) [Richardson & Domingos, 2006]

specify our problems (lacks quantifiers)

structure that solvers could exploit

Markov Logic Network: Syntax

(constraints)	C	$::=$	(H, S)			
(hard constraints)	H	$::=$	$\{ h_1, \ldots, h_n \}$,	h	$::=$	$\bigwedge_{i=1}^{n} t_i \Longrightarrow \bigvee_{i=1}^{m} t_i{}'$
(soft constraints)	S	$::=$	$\{ s_1, \ldots, s_n \}$,	s	$::=$	$w : h$

(fact)	t	$::=$	$r(a_1, \ldots, a_n)$	(ground fact)	g	$::=$	$r(c_1, \ldots, c_n)$
(argument)	a	$::=$	$v \mid c$	(input, output)	P, Q	$\subseteq$	$\mathbf{G}$

Example:

$$\forall x.\ \mathrm{path}(x, x)$$

$$\wedge \quad \forall x, y, z.\ \mathrm{path}(x, y) \wedge \mathrm{edge}(y, z) \Longrightarrow \mathrm{path}(x, z)$$

$$\wedge \quad 1.5:\ \forall x, y.\ \neg\, \mathrm{path}(x, y)$$

Datalog-like Notation

Input relations:

edge(x, y)

Hard constraints:

path(x, x).

path(x, z) :– path(x, y), edge(y, z).

Output relations:

path(x, y)

Soft constraints:

¬ path(x, y). **weight 1.5**

Example:

$$\forall x.\ \text{path}(x, x)$$

$$\wedge \quad \forall x, y, z.\ \text{path}(x, y) \wedge \text{edge}(y, z) \Longrightarrow \text{path}(x, z)$$

$$\wedge \quad 1.5{:}\ \forall x, y.\ \neg\, \text{path}(x, y)$$

Markov Logic Network: Grounding

(valuation) $\boldsymbol{\sigma} \in \mathbf{V} \to \mathbf{C}$

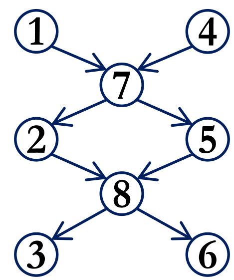
$$
\begin{aligned}
[\![(H, S)]\!] &= ([\![H]\!], [\![S]\!]) \\
[\![\{ h_1, \ldots, h_n \}]\!] &= \bigwedge_{i=1}^{n} [\![h_i]\!] \\
[\![\{ s_1, \ldots, s_n \}]\!] &= \bigwedge_{i=1}^{n} [\![s_i]\!] \\
[\![h]\!] &= \bigwedge_{\boldsymbol{\sigma}} [\![h]\!]_{\boldsymbol{\sigma}} \\
[\![w : h]\!] &= \bigwedge_{\boldsymbol{\sigma}} (w, [\![h]\!]_{\boldsymbol{\sigma}}) \\
[\![\bigwedge_{i=1}^{n} t_i \Longrightarrow \bigvee_{i=1}^{m} t_i']\!]_{\boldsymbol{\sigma}} &= \bigvee_{i=1}^{n} \neg [\![t_i]\!]_{\boldsymbol{\sigma}} \vee \bigvee_{i=1}^{m} [\![t_i']\!]_{\boldsymbol{\sigma}} \\
[\![r(a_1, \ldots, a_n)]\!]_{\boldsymbol{\sigma}} &= r(\boldsymbol{\sigma}(a_1), \ldots, \boldsymbol{\sigma}(a_n)) \\
[\![v]\!]_{\boldsymbol{\sigma}} &= \boldsymbol{\sigma}(v) \\
[\![c]\!]_{\boldsymbol{\sigma}} &= c
\end{aligned}
$$

Markov Logic Network: Semantics

- Conceptually, two steps:
  - Step 1: Ground the MLN instance
    - Substitute quantifiers by all possible valuations to constants, to yield a MaxSAT instance
  - Step 2: Solve the MaxSAT instance
    - Using off-the-shelf MaxSAT solver
- Challenge: both steps intractable for our problems
  - MaxSAT instances can comprise upto 10^30 clauses!
- Solution: Iterative lazy refinement

Markov Logic Network: Example

Input relations:
edge(x, y)
Output relations:
path(x, y)
Hard constraints:
path(x, x).
path(x, z) :- path(x, y), edge(y, z).
Soft constraints:
¬ path(x, y). **weight 1.5**



Input:
edge(1, 7), edge(4, 7), …

Grounding →

Hard clauses:

$$edge(1,7) \wedge edge(4,7) \wedge \ldots \wedge$$
$$path(1,1) \wedge path(2,2) \wedge \ldots \wedge$$
$$(path(1,1) \vee \neg path(1,1) \vee \neg edge(1,1)) \wedge$$
$$(path(1,2) \vee \neg path(1,1) \vee \neg edge(1,2)) \wedge$$
$$(path(1,2) \vee \neg path(1,2) \vee \neg edge(2,2)) \wedge$$
$$\ldots$$

Soft clauses:

$$(\neg path(1,1) \ \mathbf{weight\ 1.5}) \wedge$$
$$(\neg path(1,2) \ \mathbf{weight\ 1.5}) \wedge$$
$$\ldots$$

← **Solving**

Output:

$path(1,1) = \mathrm{T},\ path(2,2) = \mathrm{T},$
$path(1,2) = \mathrm{T},\ path(2,1) = \mathrm{F},$
$\ldots$

Landscape of Problems

Focus of this Talk

INFERENCE	Maximum A Posteriori (MAP) Inference	Marginal Inference
LEARNING	Weight Learning	Structure Learning

$$\forall x.\ \mathrm{path}(x, x)$$

$$\wedge \quad \forall x, y, z.\ \mathrm{path}(x, y) \wedge \mathrm{edge}(y, z) \implies \mathrm{path}(x, z)$$

$$\wedge \quad 1.5{:}\ \forall x, y.\ \neg\, \mathrm{path}(x, y)$$

Talk Outline

- Background
- Part I: Applications in Software Analysis
- Part II: Techniques for MaxSAT Solving
- Conclusion

Applications in Software Analysis

Abstraction Selection in Automated Verification [PLDI 2014]

Input relations:

```
edge(a, b)
```

Output relations:

```
path(a, b)
```

Constraints:

```
path(a, a).
path(a, c) :- path(a, b), edge(b, c).
```

Alarm Classification in Static Bug Detection [FSE 2015]

Alarm Resolution in Interactive Verification [OOPSLA 2017]

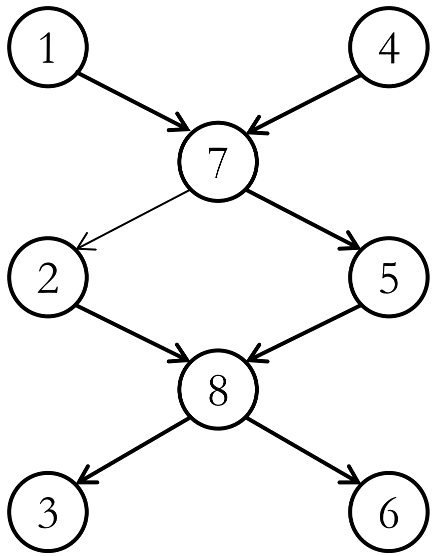
Overview of Applications

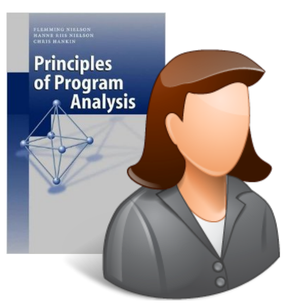
- Abstraction Selection [PLDI 2014]
- Alarm Classification [FSE 2015]
- Alarm Resolution [OOSPLA 2017]

An Example: Pointer Analysis

```
f(){                                  g(){
   v1 = new ...                          v4 = new ...
   v2 = id1(v1)                          v5 = id1(v4)
   v3 = id2(v2)                          v6 = id2(v5)
✗  assert(v3 != v1) q1               ✓  assert(v6 != v1) q2
}                                     }

id1(v){ return v }                    id2(v){ return v }
```

Pointer Analysis via Graph Reachability





Analysis Writer
(Alice)

```
f(){                          g(){
 v1 = new ...                  v4 = new ...
 v2 = id1(v1)                  v5 = id1(v4)
 v3 = id2(v2)                  v6 = id2(v5)
 assert(v3!=v1) q1             assert(v6!=v1) q2
}                             }
id1(v){ return v }            id2(v){ return v }
```

`assert(v6!=v1)` holds if there is no path from **1** to **6**.

`assert(v3!=v1)` holds if there is no path from **1** to **3**.

Analysis Specification in Datalog

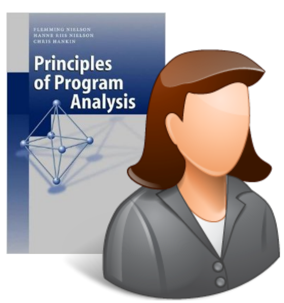
Input relations:
edge(a, b)

Output relations:
path(a, b)

Constraints:

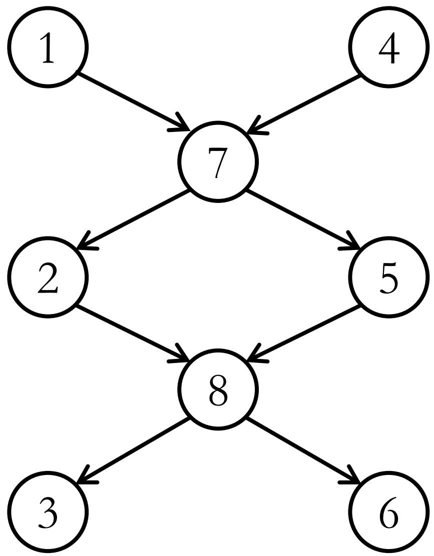
```
path(a, a).
path(a, c) :- path(a, b), edge(b, c).
```

If
there is a path from **a** to **b**, and
there is an edge from **b** to **c**,
then
there is a path from **a** to **c.**



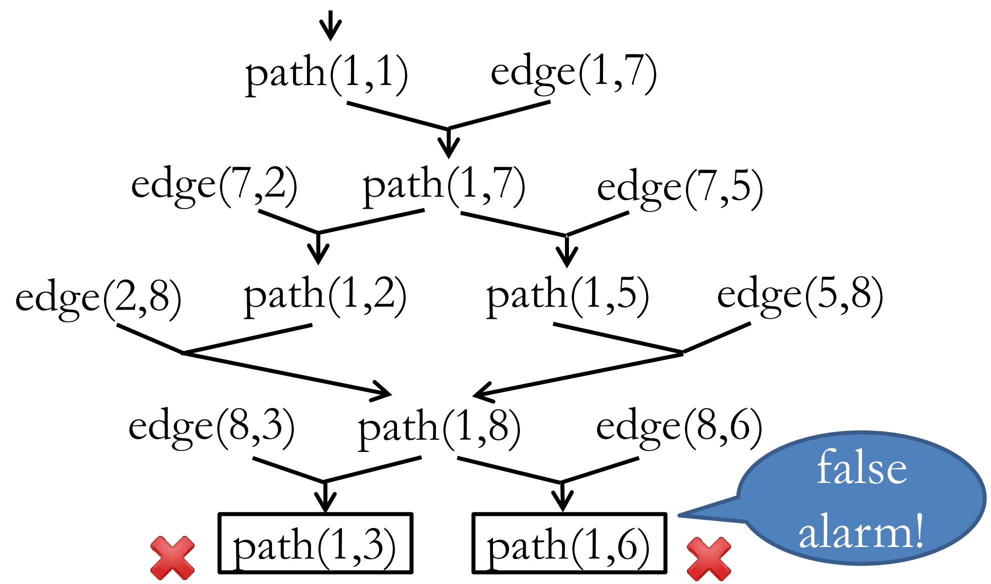
Analysis Writer (Alice)

Analysis Evaluation in Datalog

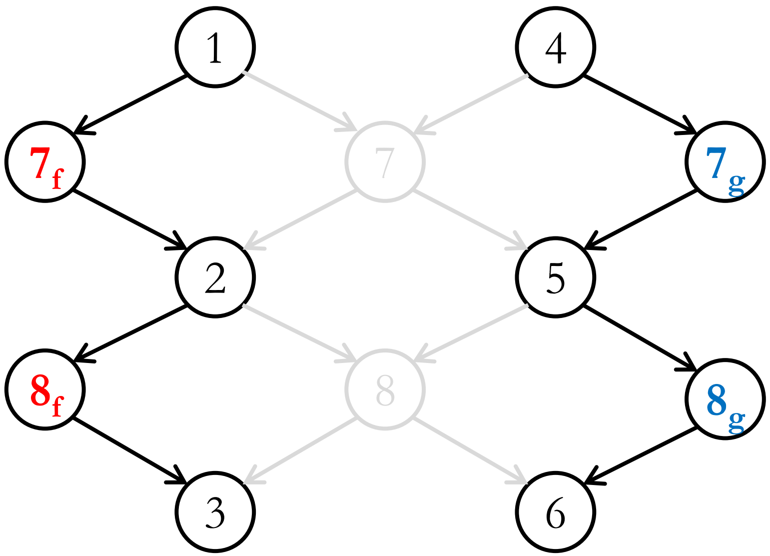


Analysis User (Bob)

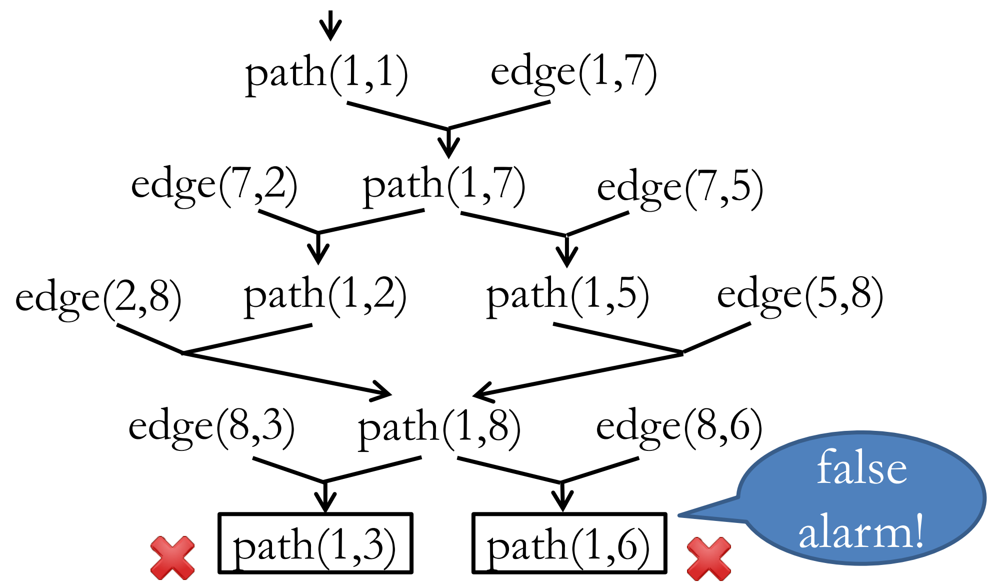
```
f(){                          g(){
 v1 = new ...                  v4 = new ...
 v2 = id1(v1)                  v5 = id1(v4)
 v3 = id2(v2)                  v6 = id2(v5)
 assert(v3!=v1)  q1            assert(v6!=v1)  q2
}                             }

id1(v){ return v }            id2(v){ return v }
```



A More Precise Abstraction

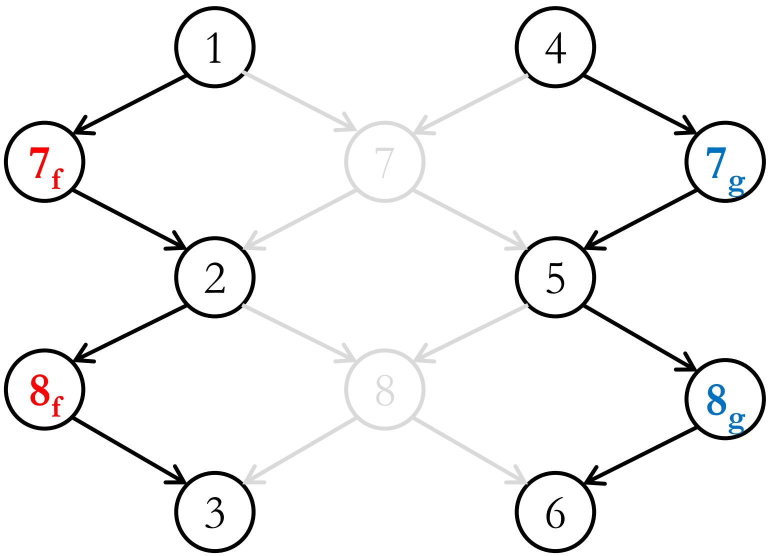



```
f(){                              g(){
 v1 = new ...                      v4 = new ...
 v2 = id1(v1)                      v5 = id1(v4)
 v3 = id2(v2)                      v6 = id2(v5)
 assert(v3!=v1) q1                 assert(v6!=v1) q2
}                                 }
id1(v){ return v }                id2(v){ return v }
```



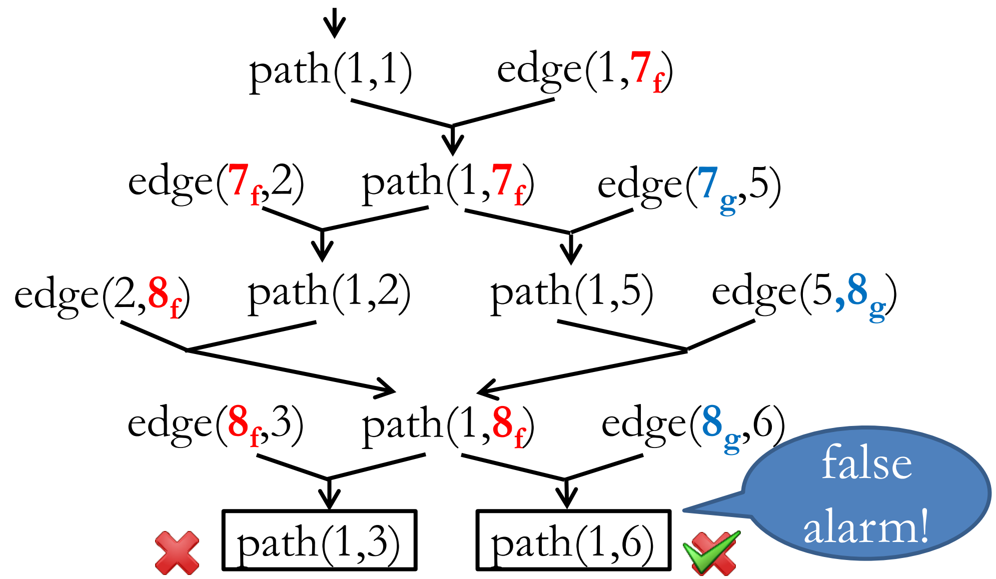
Analysis User (Bob)

A More Precise Abstraction

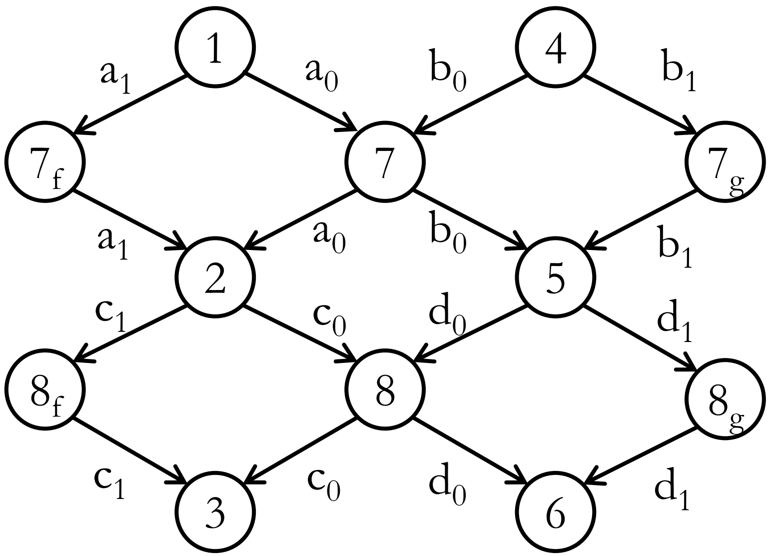



```
f(){                              g(){
 v1 = new ...                      v4 = new ...
 v2 = id1(v1)                      v5 = id1(v4)
 v3 = id2(v2)                      v6 = id2(v5)
 assert(v3!=v1) q1                 assert(v6!=v1) q2
}                                 }
id1(v){ return v }                id2(v){ return v }
```

Analysis User (Bob)



Abstraction Refinement



Input relations: edge(a, b), abs(n)

Output relations: path(a, b)

Constraints:

path(a, a).

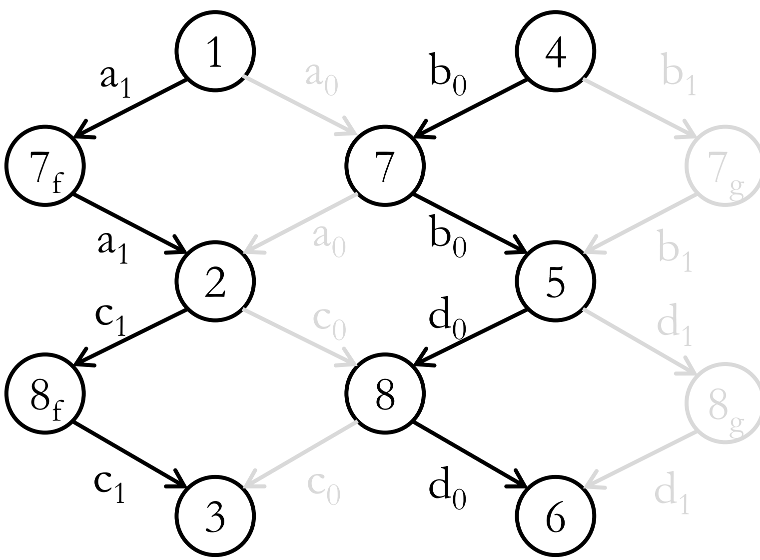
path(a, c) :- path(a, b), edge(b, c).n), abs(n).

$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$, $abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

16 possible abstractions in total

Query Tuple	Original Query
q_1: path(1, 3)	`assert(v3!=v1)`
q_2: path(1, 6)	`assert(v6!=v1)`

Desired Result



Input relations: edge(a, b), abs(n)

Output relations: path(a, b)

Constraints:

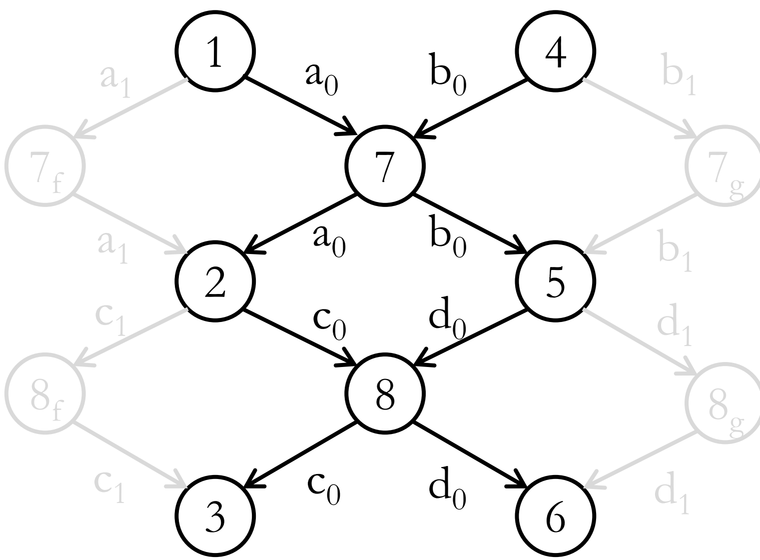
path(a, a).

path(a, c) :- path(a, b), edge(b, c, n), abs(n).

$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$,
$abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

Query	Answer
q_1: path(1, 3)	✗ Impossibility
q_2: path(1, 6)	✓ $a_1b_0c_1d_0$

Iteration 1

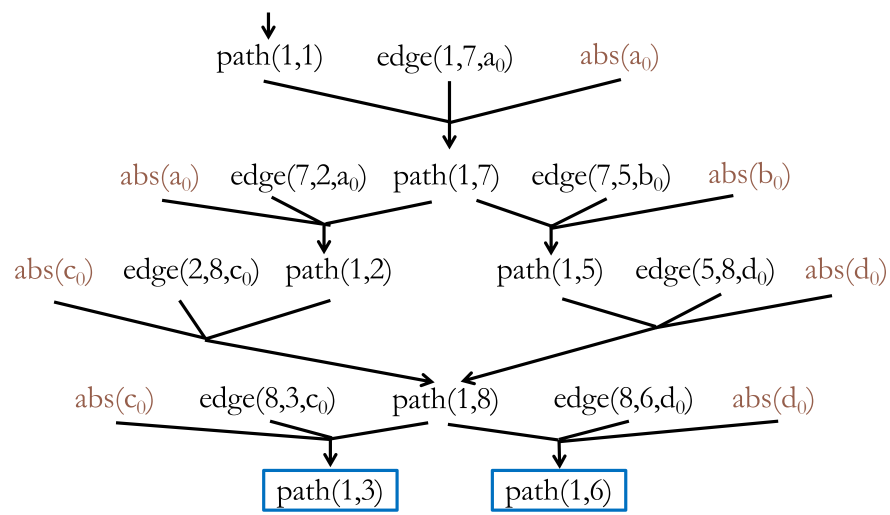


Input relations:
edge(a, b), abs(n)

Output relations:
path(a, b)

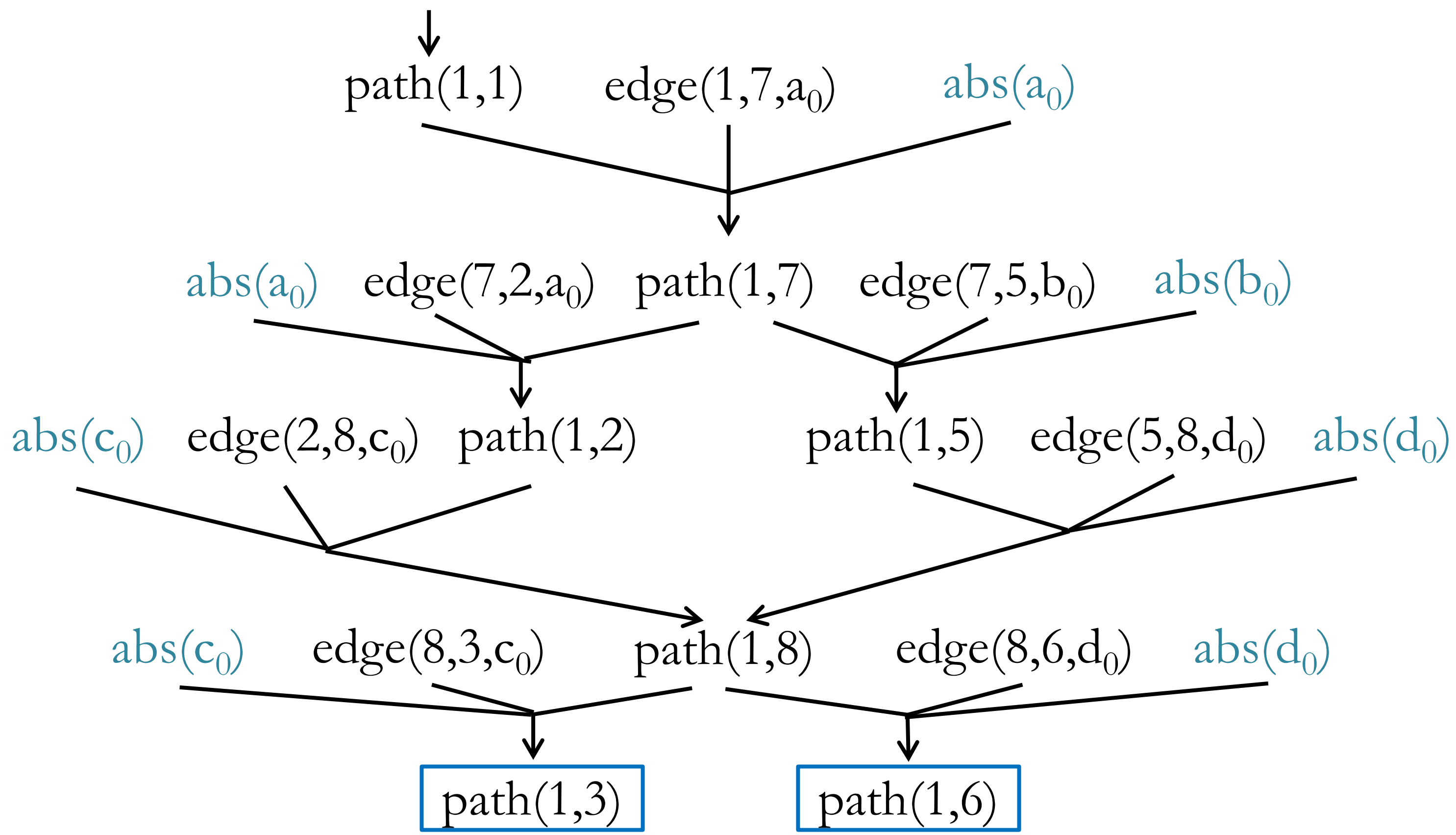
Constraints:
path(a, a).
path(a, c) :- path(a, b), edge(b, c, n), abs(n).

$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$,
$abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

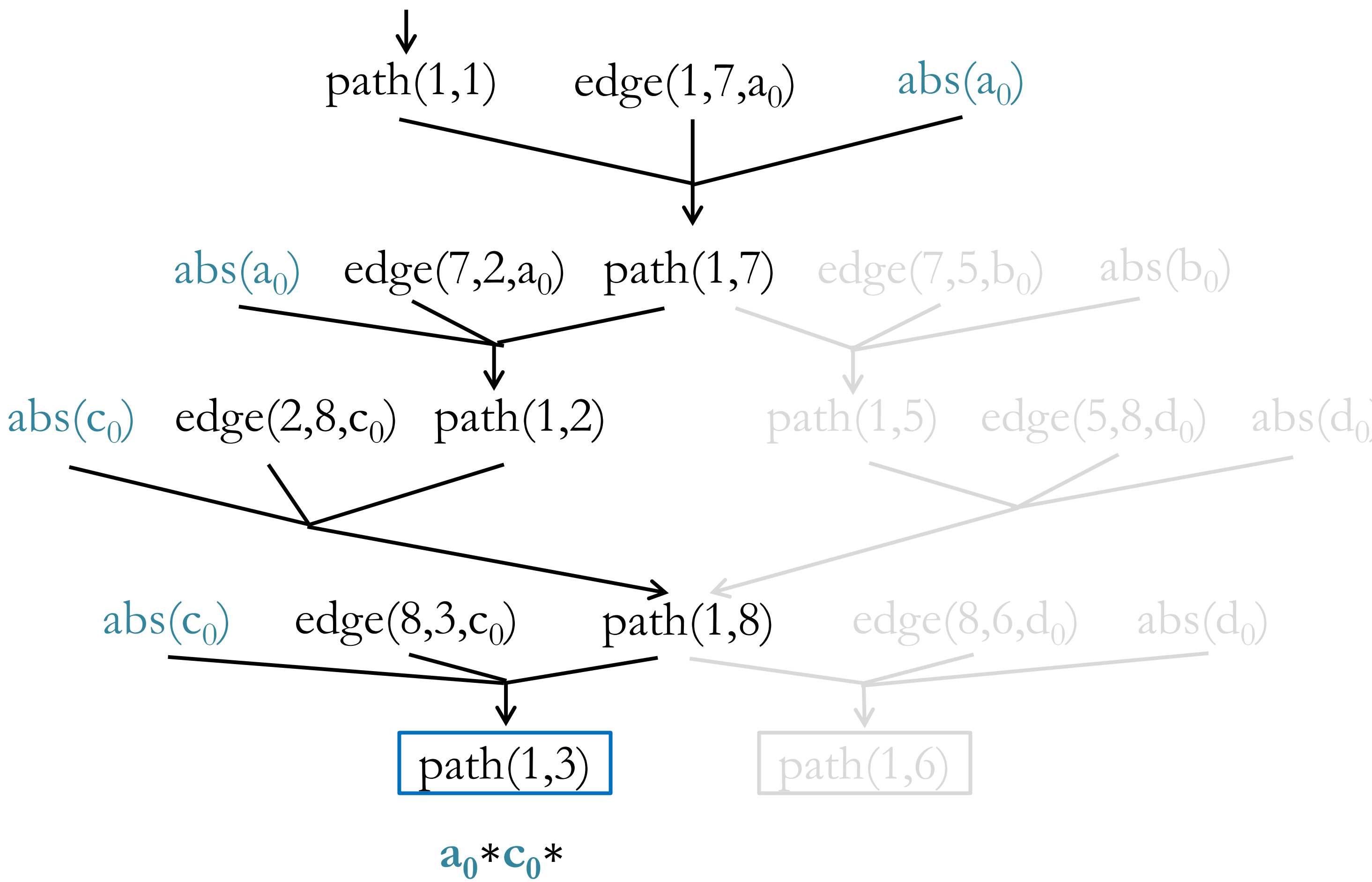


Query	Eliminated Abstractions
q_1: path(1, 3)	
q_2: path(1, 6)	

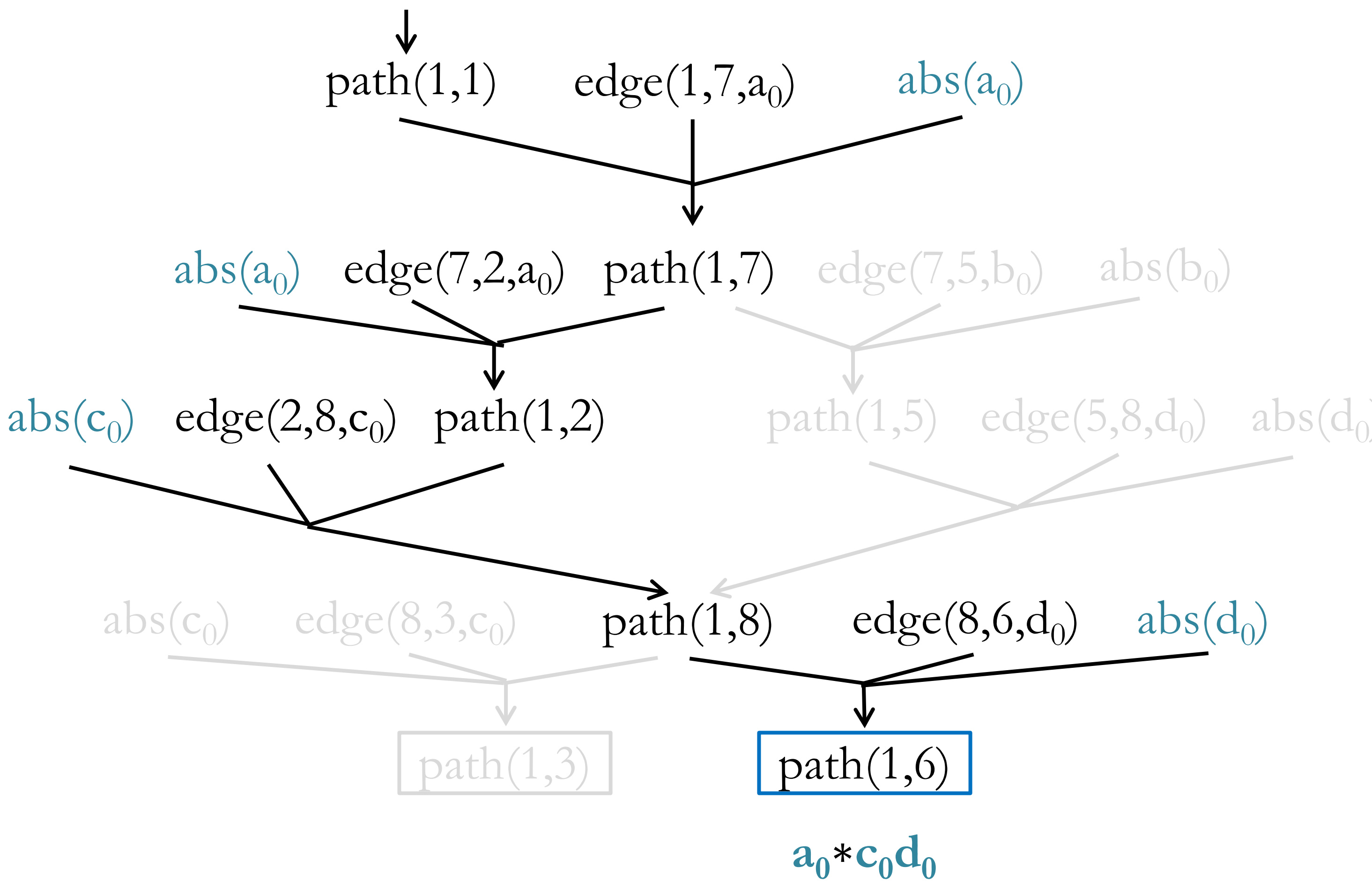
Iteration 1 - Derivation Graph



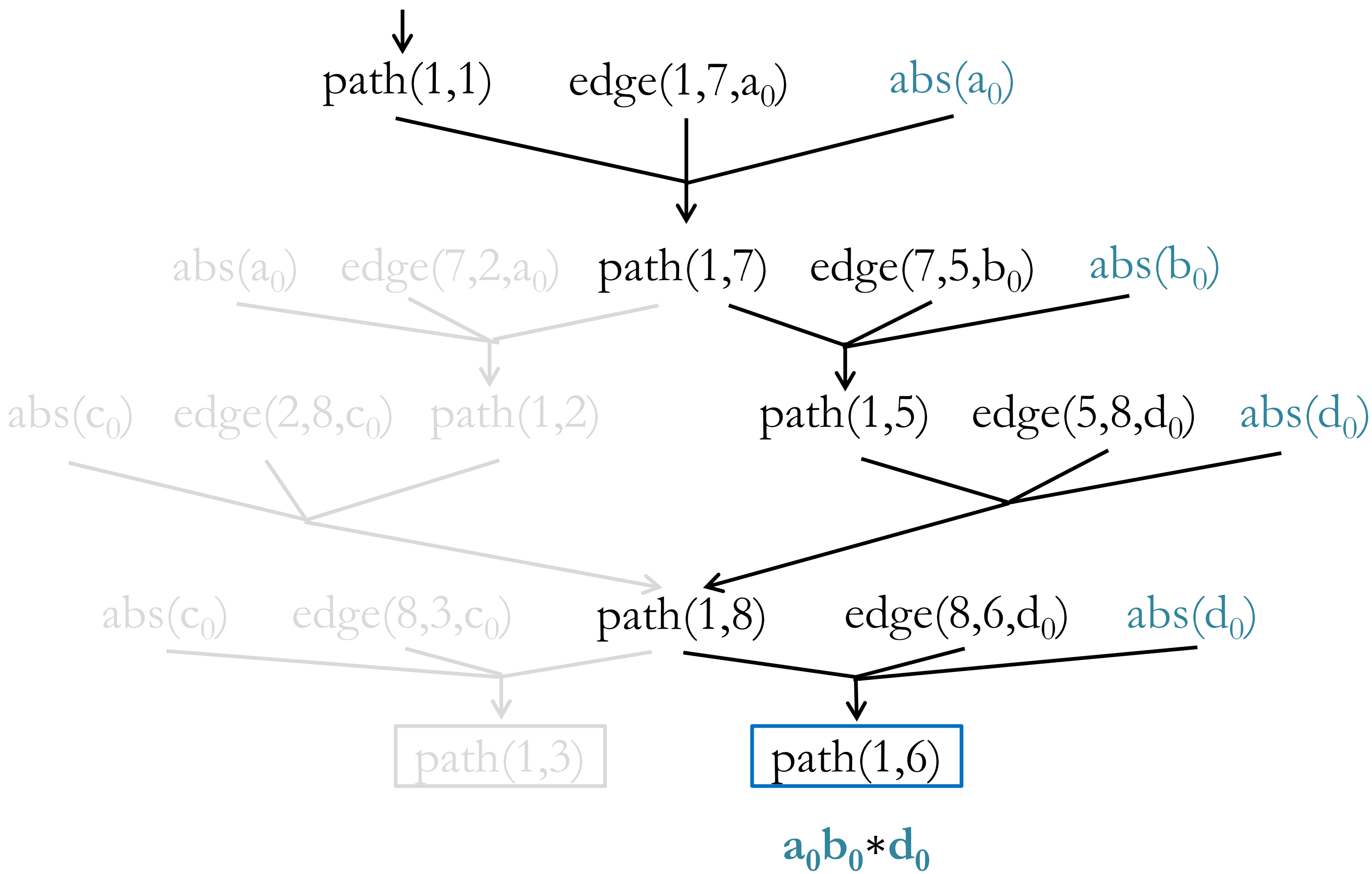
Iteration 1 - Derivation Graph



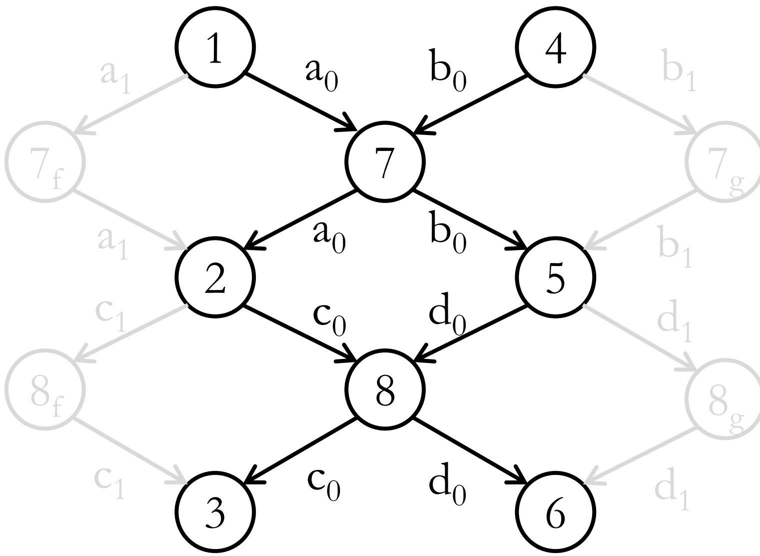
Iteration 1 - Derivation Graph

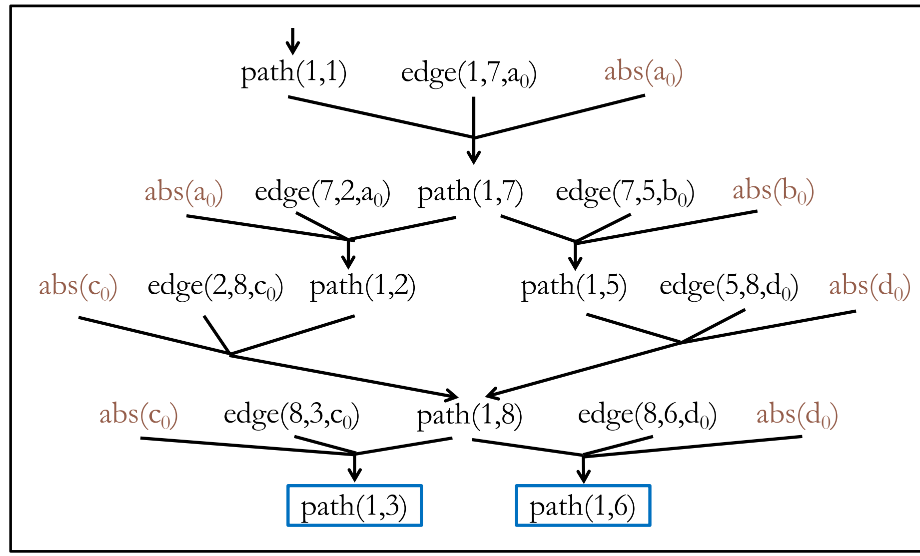


Iteration 1 - Derivation Graph



Iteration 1 - Derivation Graph

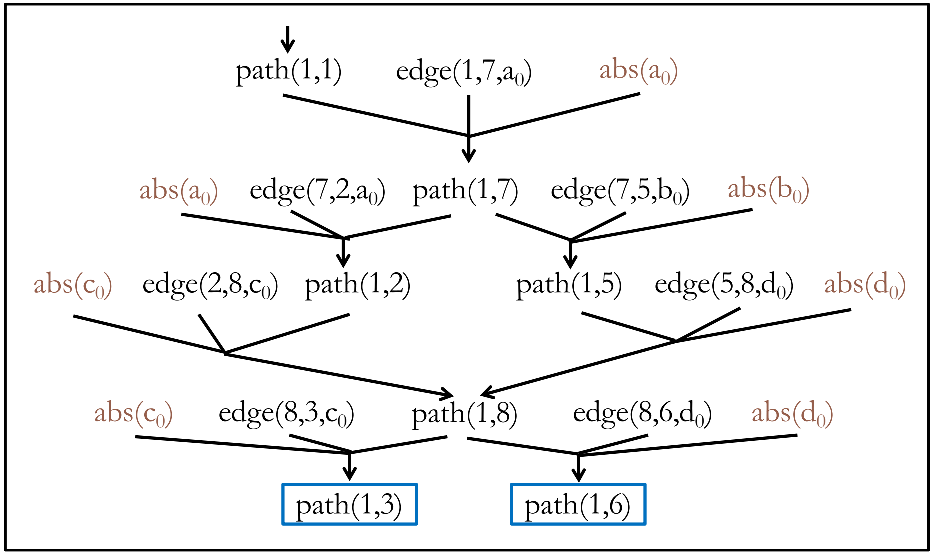




Query	Eliminated Abstractions
q_1: path(1, 3)	$a_0{*}c_0{*}$ **(4/16)**
q_2: path(1, 6)	$a_0{*}c_0d_0$, $a_0b_0{*}d_0$ **(3/16)**

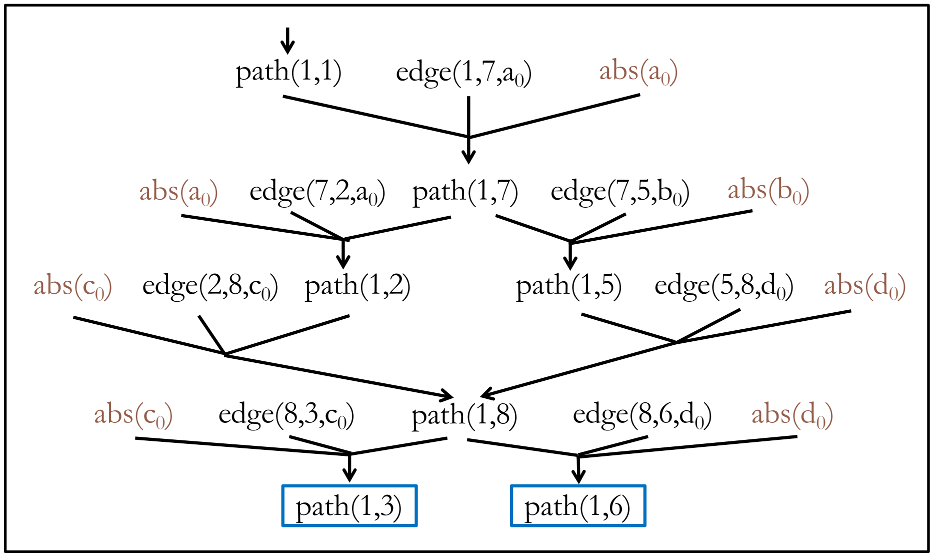
$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$,
$abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

Encoding in MaxSAT



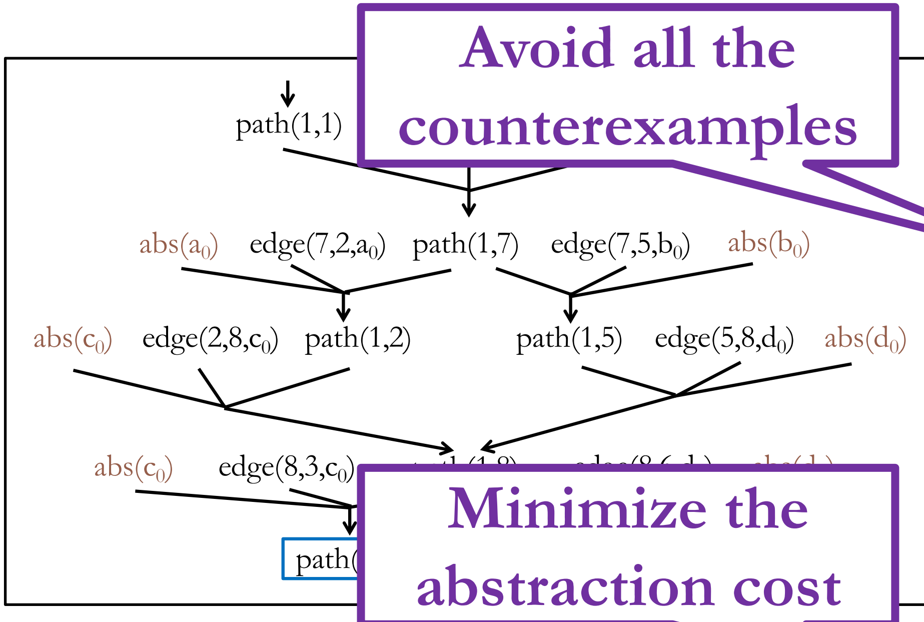
$\text{abs}(a_0) \oplus \text{abs}(a_1)$, $\text{abs}(b_0) \oplus \text{abs}(b_1)$,
$\text{abs}(c_0) \oplus \text{abs}(c_1)$, $\text{abs}(d_0) \oplus \text{abs}(d_1)$.

Encoding in MaxSAT



$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$,
$abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

Encoding in MaxSAT



Hard constraints:

$$path(1,1) \wedge$$
$$(path(1,7) \vee \neg path(1,1) \vee \neg abs(a_0)) \wedge$$
$$(path(1,2) \vee \neg path(1,7) \vee \neg abs(a_0)) \wedge$$
$$(path(1,8) \vee \neg path(1,2) \vee \neg abs(c_0)) \wedge$$
$$(path(1,3) \vee \neg path(1,8) \vee \neg abs(c_0)) \wedge$$
$$(path(1,5) \vee \neg path(1,7) \vee \neg abs(b_0)) \wedge$$
$$(path(1,8) \vee \neg path(1,5) \vee \neg abs(d_0)) \wedge$$
$$(path(1,6) \vee \neg path(1,8) \vee \neg abs(d_0))$$

Soft constraints:

$$(abs(a_0)\ \textbf{weight 1}) \wedge$$
$$(abs(b_0)\ \textbf{weight 1}) \wedge$$
$$(abs(c_0)\ \textbf{weight 1}) \wedge$$
$$(abs(d_0)\ \textbf{weight 1}) \wedge$$
$$(\neg path(1,3)\ \textbf{weight 5}) \wedge$$
$$(\neg path(1,6)\ \textbf{weight 5})$$

$abs(a_0) \oplus abs(a_1)$, $abs(b_0) \oplus abs(b_1)$,
$abs(c_0) \oplus abs(c_1)$, $abs(d_0) \oplus abs(d_1)$.

Encoding in MaxSAT

Solution:

$path(1,1) = \text{true}$, $path(1,2) = \text{false}$,
$path(1,3) = \text{false}$, $path(1,5) = \text{false}$,
$path(1,6) = \text{false}$, $path(1,7) = \text{false}$,
$path(1,8) = \text{false}$,
$abs(a_0) = \text{false}$, $abs(b_0) = \text{true}$,
$abs(c_0) = \text{true}$, $abs(d_0) = \text{true}$.

$a_1 b_0 c_0 d_0$

Query	Eliminated Abstractions	
q_1: path(1, 3)	$a_0 * c_0 *$	**(4/16)**
q_2: path(1, 6)	$a_0 * c_0 d_0$, $a_0 b_0 * d_0$	**(3/16)**

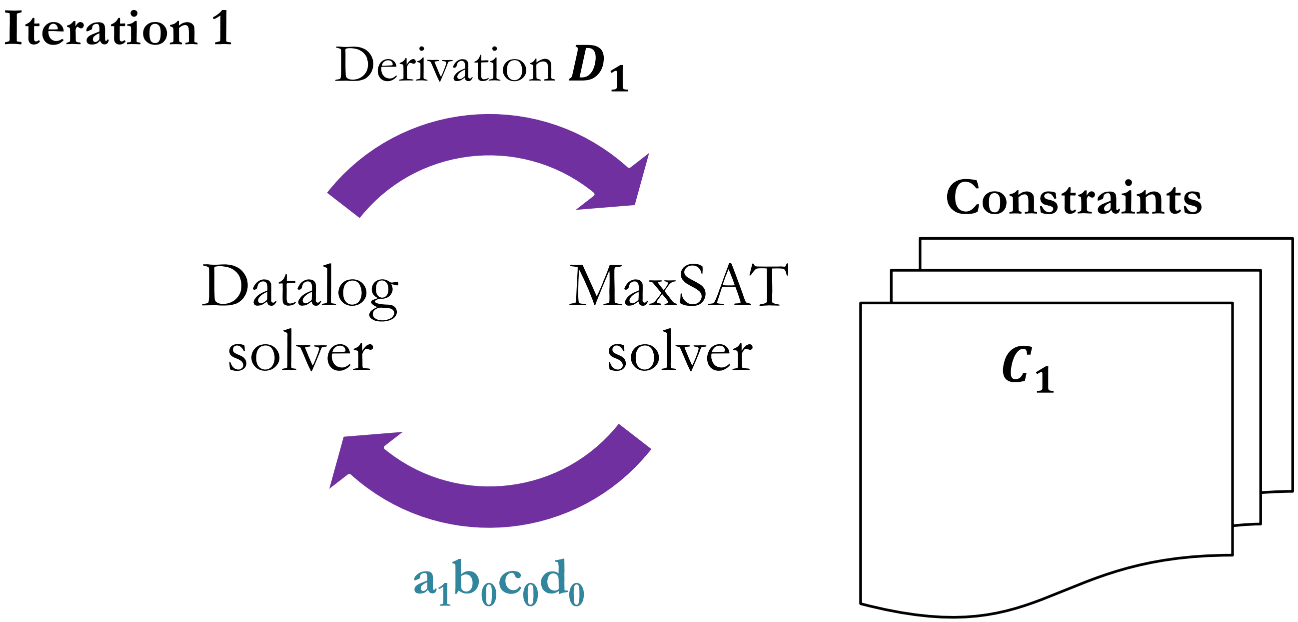
Hard constraints:

$$path(1,1) \wedge$$
$$(path(1,7) \vee \neg path(1,1) \vee \neg abs(a_0)) \wedge$$
$$(path(1,2) \vee \neg path(1,7) \vee \neg abs(a_0)) \wedge$$
$$(path(1,8) \vee \neg path(1,2) \vee \neg abs(c_0)) \wedge$$
$$(path(1,3) \vee \neg path(1,8) \vee \neg abs(c_0)) \wedge$$
$$(path(1,5) \vee \neg path(1,7) \vee \neg abs(b_0)) \wedge$$
$$(path(1,8) \vee \neg path(1,5) \vee \neg abs(d_0)) \wedge$$
$$(path(1,6) \vee \neg path(1,8) \vee \neg abs(d_0))$$

Soft constraints:

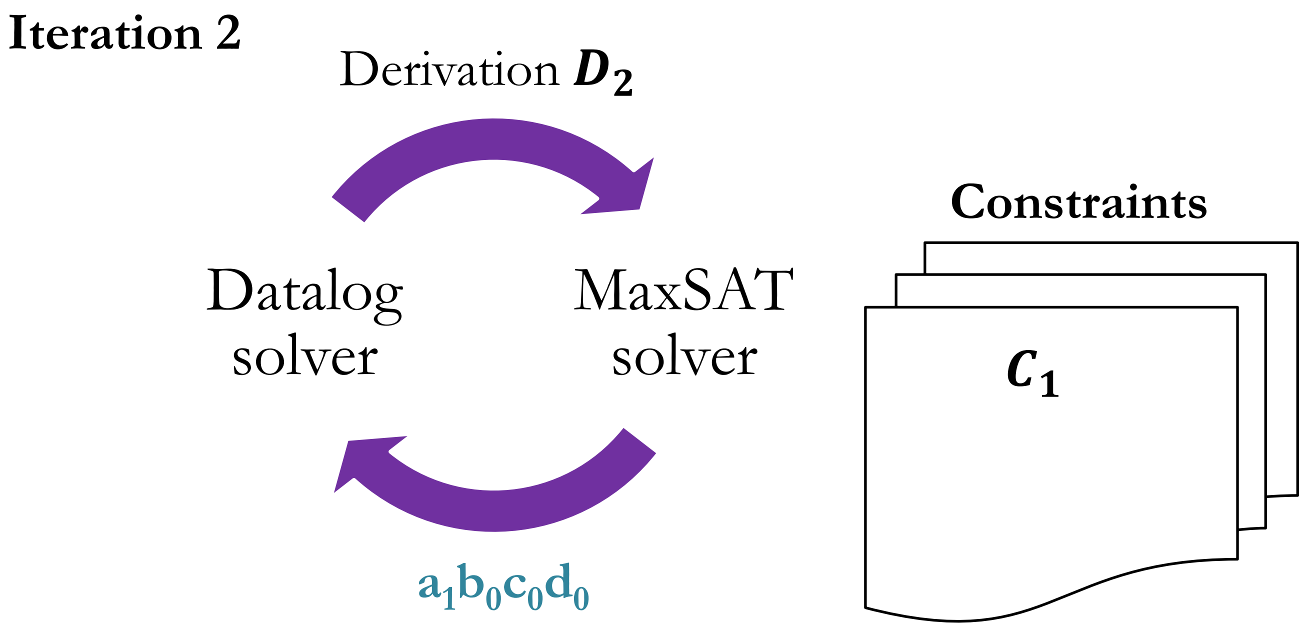
$$(abs(a_0)\ \textbf{weight 1}) \wedge$$
$$(abs(b_0)\ \textbf{weight 1}) \wedge$$
$$(abs(c_0)\ \textbf{weight 1}) \wedge$$
$$(abs(d_0)\ \textbf{weight 1}) \wedge$$
$$(\neg path(1,3)\ \textbf{weight 5}) \wedge$$
$$(\neg path(1,6)\ \textbf{weight 5})$$

Iteration 2 and Beyond



Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		a_0*c_0* **(4/16)**
q_2: path(1, 6)		$a_0*c_0d_0$, $a_0b_0*d_0$ **(3/16)**

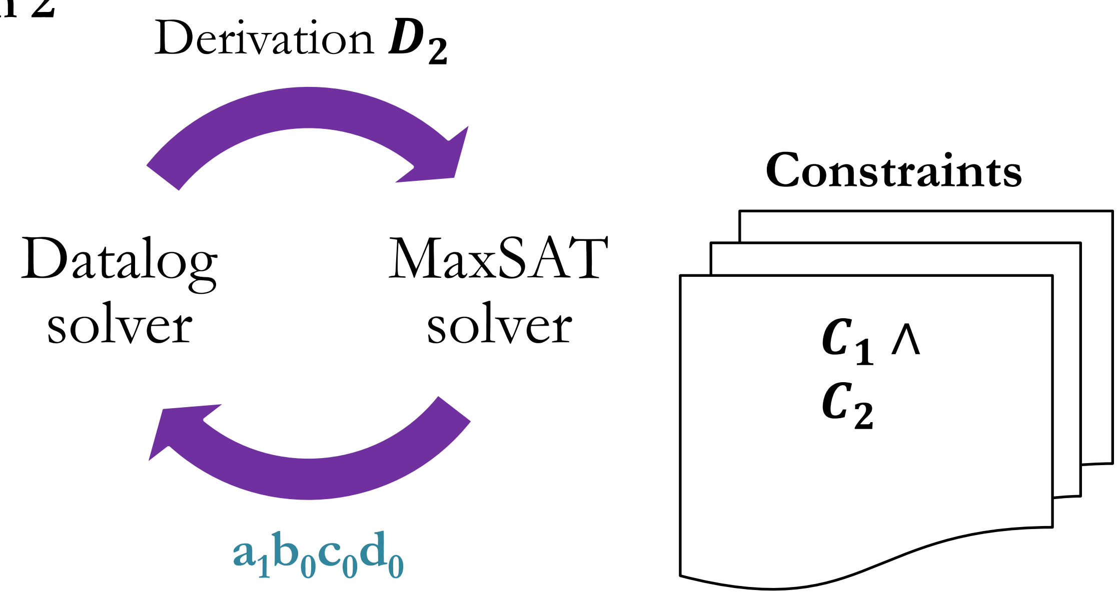
Iteration 2 and Beyond



Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		$a_0 * c_0 *$ **(4/16)**
q_2: path(1, 6)		$a_0 * c_0 d_0$, $a_0 b_0 * d_0$ **(3/16)**

Iteration 2 and Beyond

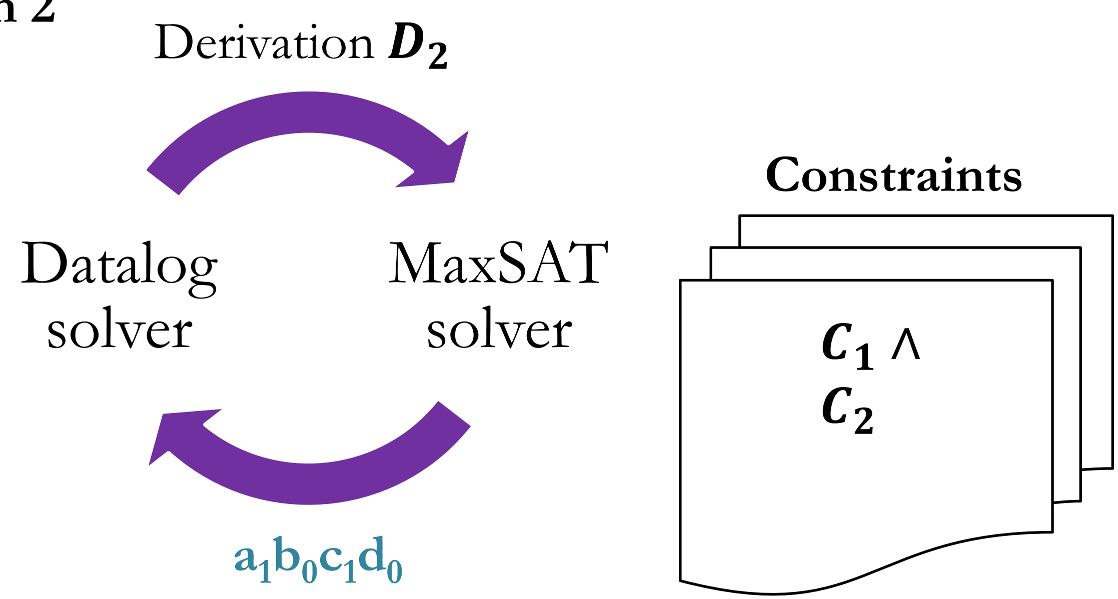
Iteration 2



Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		$a_0{*}c_0{*}$ **(4/16)**
q_2: path(1, 6)		$a_0{*}c_0d_0$, $a_0b_0{*}d_0$ **(3/16)**

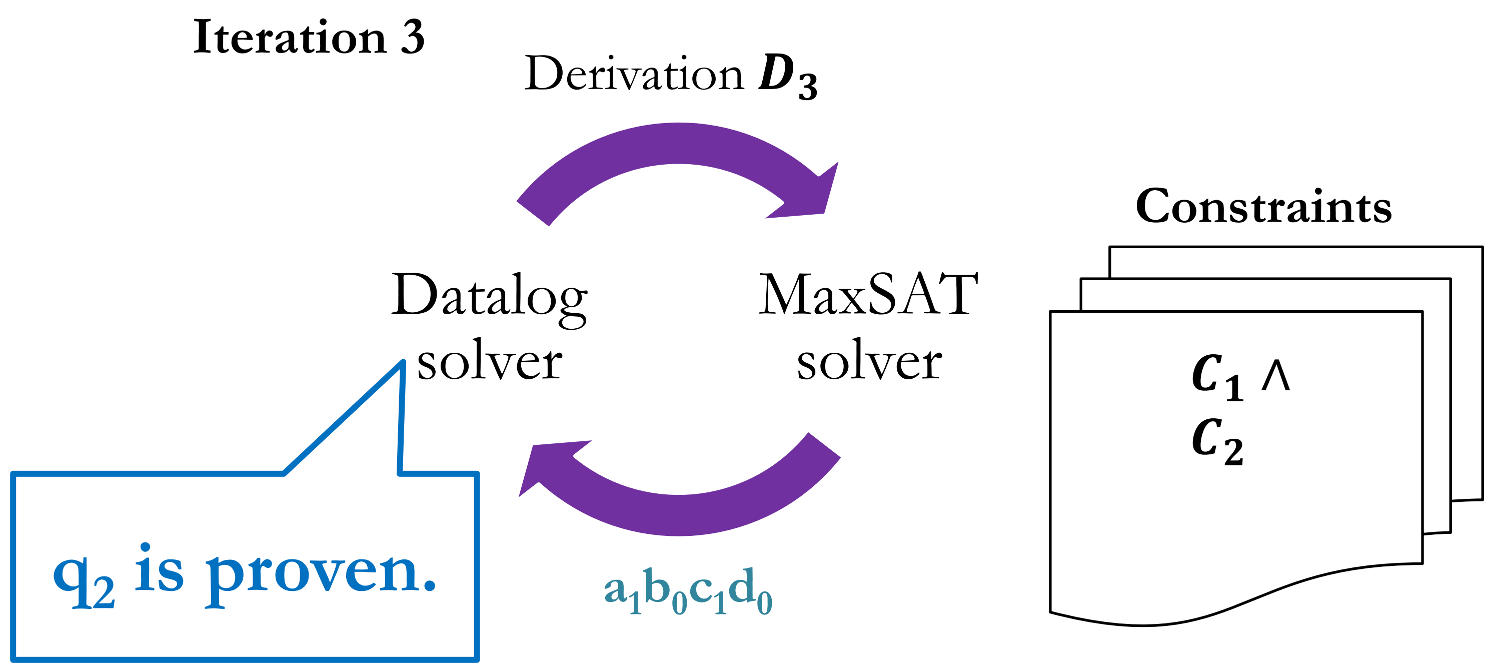
Iteration 2 and Beyond

Iteration 2



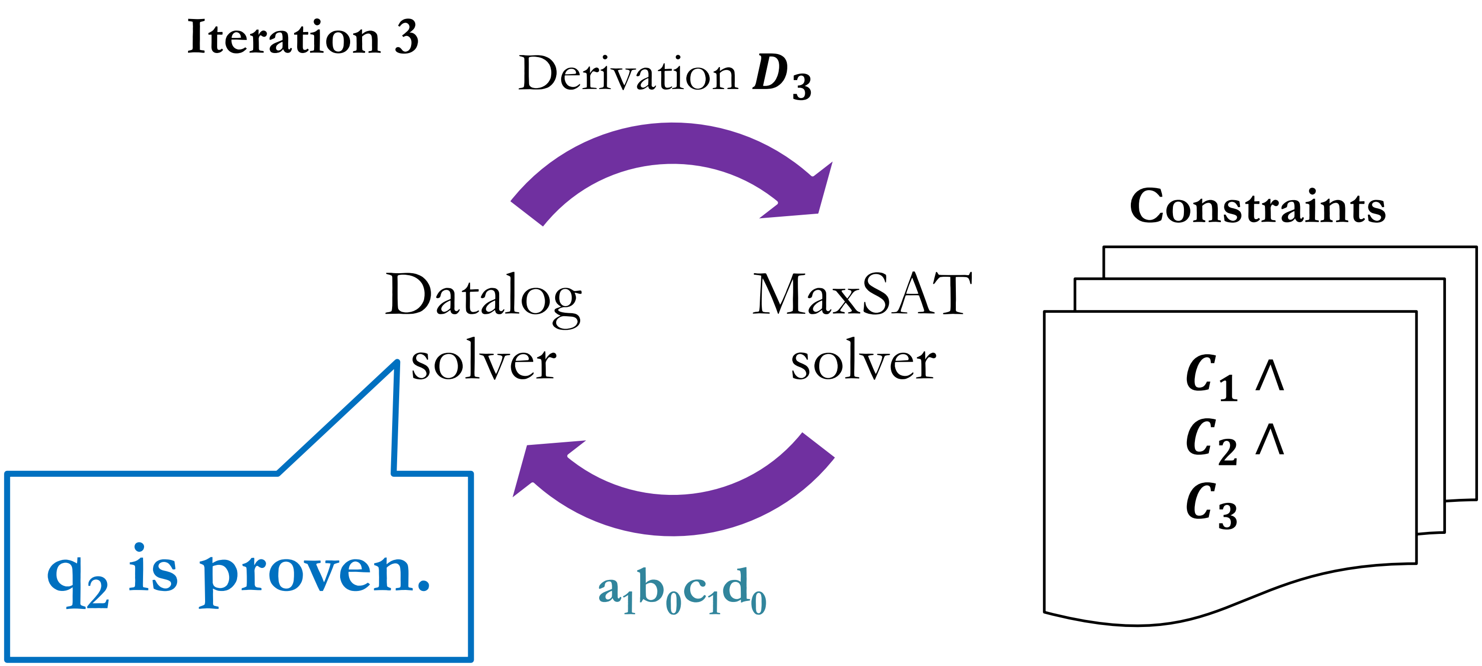
Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		$a_0{*}c_0{*}$, $a_1{*}c_0{*}$ **(8/16)**
q_2: path(1, 6)		$a_0{*}c_0d_0$, $a_0b_0{*}d_0$, $a_1{*}c_0d_0$ **(5/16)**

Iteration 2 and Beyond



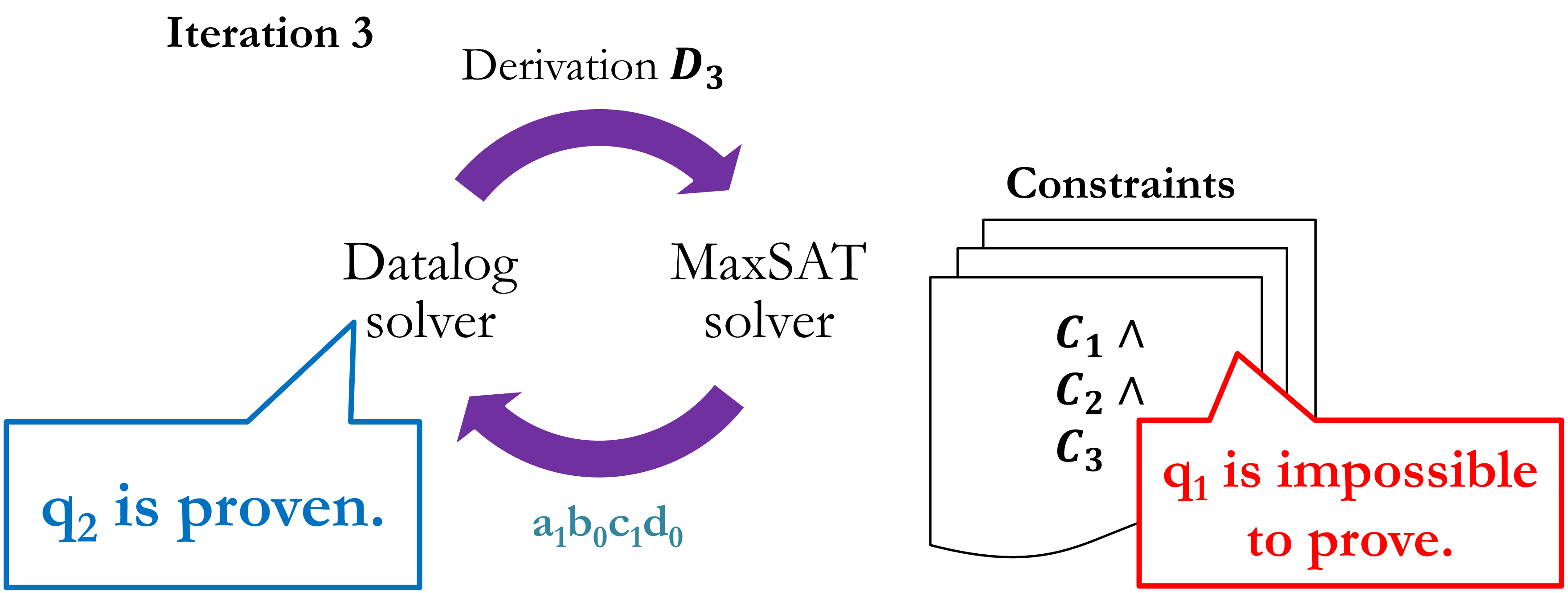
Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		a_0*c_0*, a_1*c_0* **(8/16)**
q_2: path(1, 6)	✓ $a_1b_0c_1d_0$	$a_0*c_0d_0$, $a_0b_0*d_0$, $a_1*c_0d_0$ **(5/16)**

Iteration 2 and Beyond



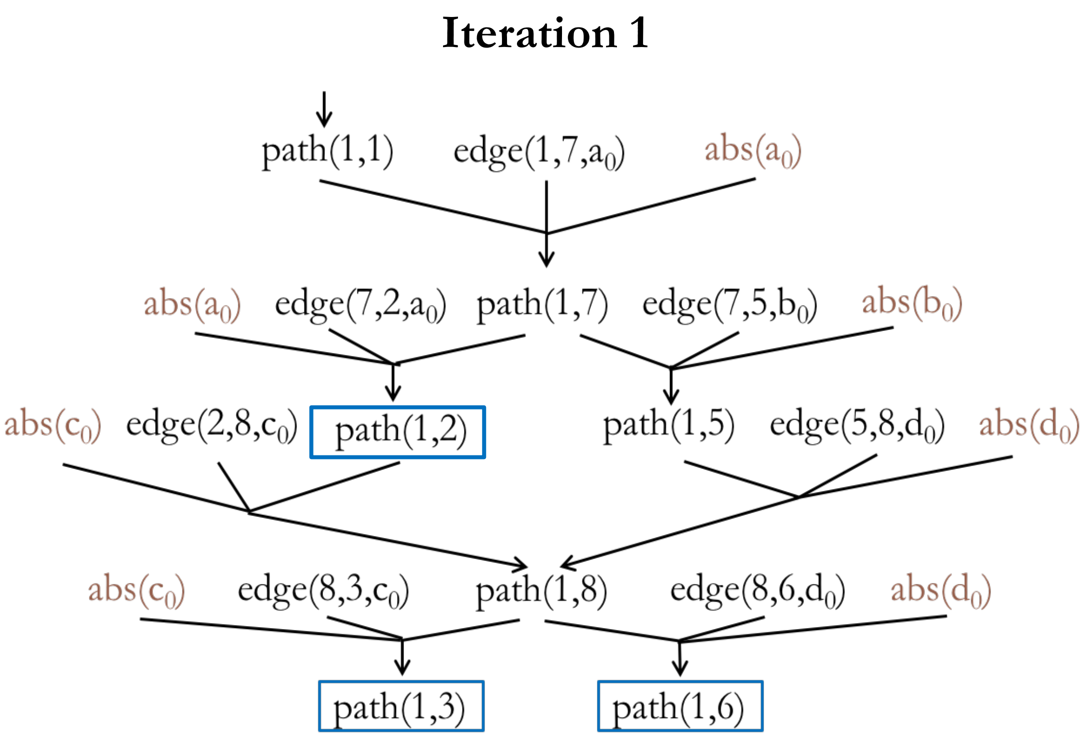
Query	Answer	Eliminated Abstractions
q_1: path(1, 3)		a_0*c_0*, a_1*c_0* **(8/16)**
q_2: path(1, 6)	✓ $a_1b_0c_1d_0$	$a_0*c_0d_0$, $a_0b_0*d_0$, $a_1*c_0d_0$ **(5/16)**

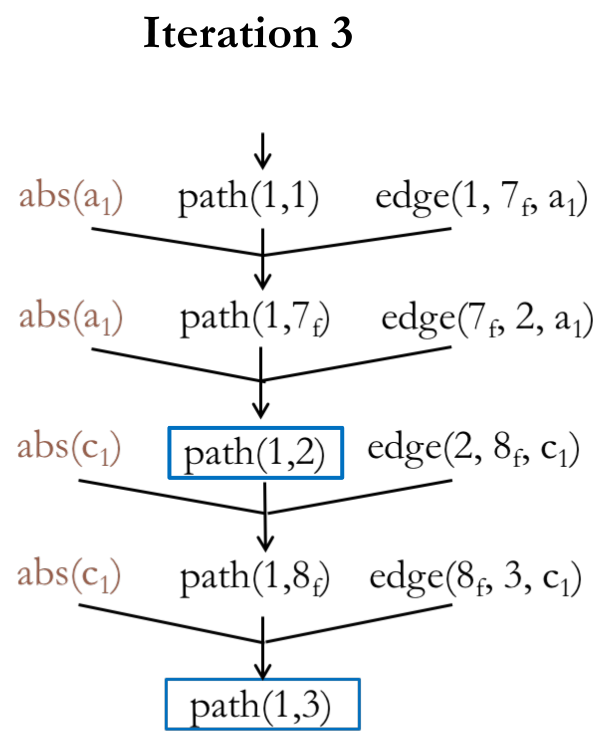
Iteration 2 and Beyond



Query	Answer	Eliminated Abstractions
q_1: path(1, 3)	✗ Impossibility	$a_0{*}c_0{*}$, $a_1{*}c_0{*}$, $a_0{*}c_1{*}$, $a_1{*}c_1{*}$ (16/16)
q_2: path(1, 6)	✓ $a_1b_0c_1d_0$	$a_0{*}c_0d_0$, $a_0b_0{*}d_0$, $a_1{*}c_0d_0$ (5/16)

Mixing Counterexamples

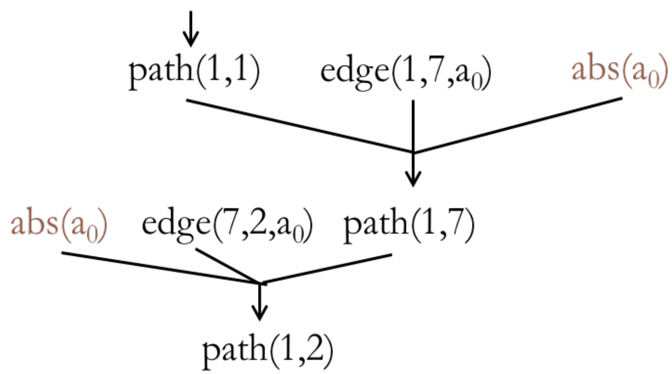


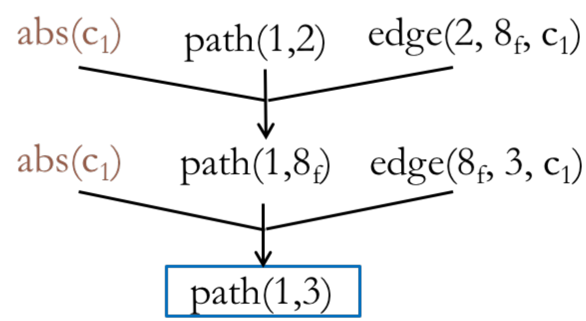


Eliminated Abstractions: $a_0{*}c_0{*}$ $a_1{*}c_1{*}$

Mixing Counterexamples

Iteration 1 | **Mixed!** | **Iteration 3**





Eliminated Abstractions: $a_0{*}c_0{*}$ $a_0{*}c_1{*}$ $a_1{*}c_1{*}$

Experimental Setup

- Implemented using off-the-shelf solvers
  - Datalog: bddbddb
  - MaxSAT: MiFuMaX

- Applied to two analyses that are challenging to scale
  - **k-object-sensitive pointer analysis**
    - flow-insensitive, weak updates, cloning-based
  - **type-state analysis**
    - flow-sensitive, strong updates, summary-based

76 rules, 50 input relations, 42 output relations

52 rules, 33 input relations, 14 output relations

- Evaluated on 8 Java programs (250-450 KLOC each)

Pointer Analysis Results

	queries			abstraction size		iterations
	total	resolved				
		current	baseline	final	max	
toba-s	7	7	0	170	18K	10
javasrc-p	46	46	0	470	18K	13
weblech	5	5	2	140	31K	10
hedc	47	47	6	730	29K	18
antlr	143	143	5	970	29K	15
luindex	138	138	67	1K	40K	26
lusearch	322	322	29	1K	39K	17
schroeder-m	51	51	25	450	58K	15

4-object-sensitivity < 50%

< 3% of max

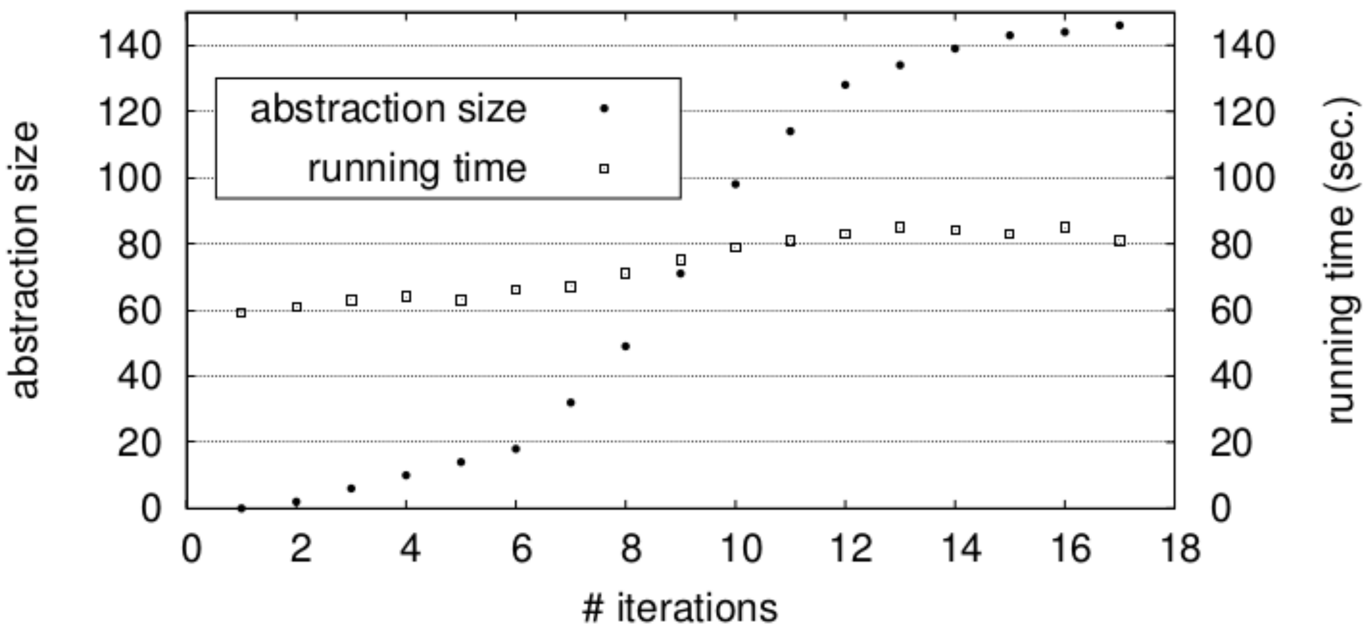
Performance of Datalog Solver

k = 4, 3h28m

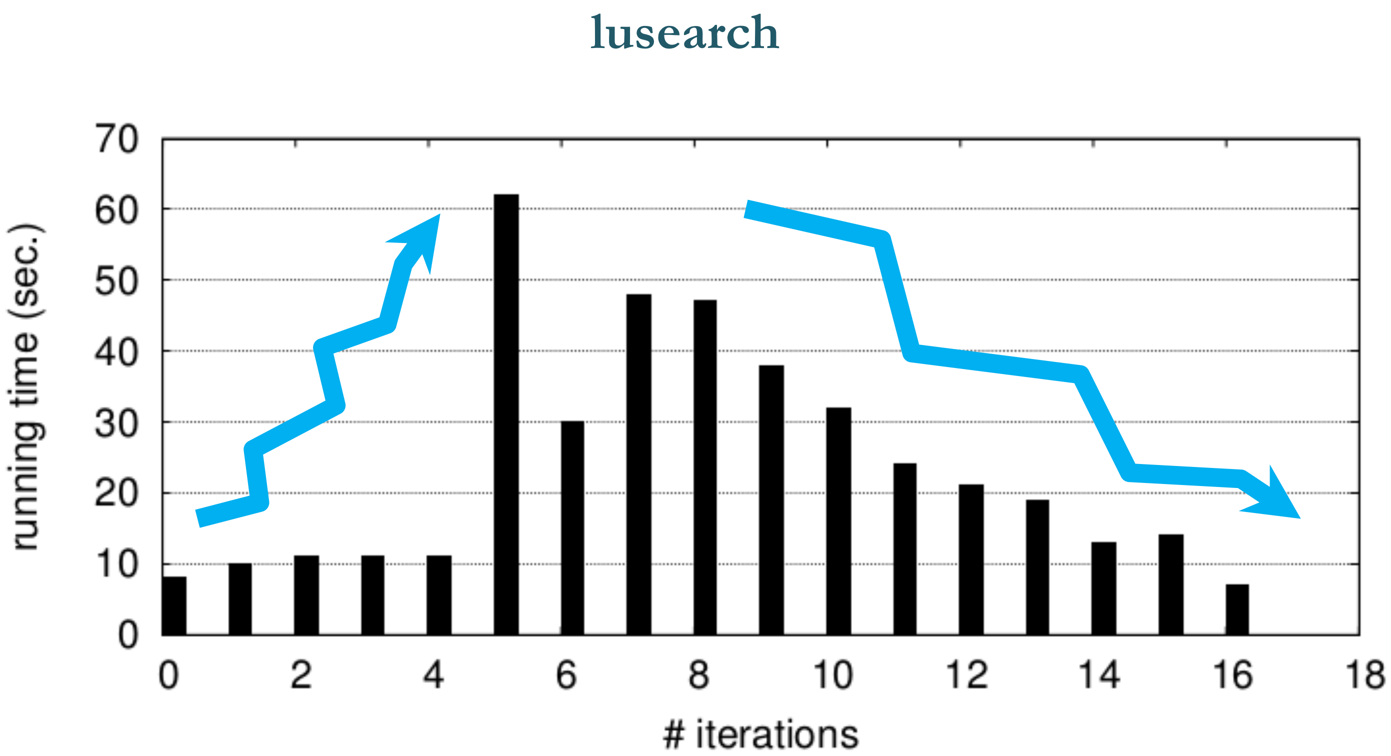
Baseline k = 3, 590s

k = 2, 214s **lusearch**

k = 1, 153s



Performance of MaxSAT Solver



Statistics of MaxSAT Formulae

	pointer analysis	
	variables	**clauses**
toba-s	0.7M	1.5M
javasrc-p	0.5M	0.9M
weblech	1.6M	3.3M
hedc	1.2M	2.7M
antlr	3.6M	6.9M
luindex	2.4M	5.6M
lusearch	2.1M	5.0M
schroeder-m	6.7M	23.7M

Overview of Applications

- Abstraction Selection [PLDI 2014]
- Alarm Classification [FSE 2015]
- Alarm Resolution [OOSPLA 2017]

How Do We Go From This …

Detected Races	
R1: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 9	org.apache.ftpserver.RequestHandler: 18
R2: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 17	org.apache.ftpserver.RequestHandler: 18
R3: Race on field org.apache.ftpserver.RequestHandler.m_writer	
org.apache.ftpserver.RequestHandler: 19	org.apache.ftpserver.RequestHandler: 20
R4: Race on field org.apache.ftpserver.RequestHandler.m_reader	
org.apache.ftpserver.RequestHandler: 21	org.apache.ftpserver.RequestHandler: 22
R5: Race on field org.apache.ftpserver.RequestHandler.m_controlSocket	
org.apache.ftpserver.RequestHandler: 23	org.apache.ftpserver.RequestHandler: 24

Eliminated Races	
E1: Race on field org.apache.ftpserver.RequestHandler. m_isConnectionClosed	
org.apache.ftpserver.RequestHandler: 13	org.apache.ftpserver.RequestHandler: 15

… To This?

Detected Races	
R1: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 9	org.apache.ftpserver.RequestHandler: 18
R2: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 17	org.apache.ftpserver.RequestHandler: 18
R3: Race on field org.apache.ftpserver.RequestHandler.m_writer	
org.apache.ftpserver.RequestHandler: 19	org.apache.ftpserver.RequestHandler: 20
R4: Race on field org.apache.ftpserver.RequestHandler.m_reader	
org.apache.ftpserver.RequestHandler: 21	org.apache.ftpserver.RequestHandler: 22
R5: Race on field org.apache.ftpserver.RequestHandler.m_controlSocket	
org.apache.ftpserver.RequestHandler: 23	org.apache.ftpserver.RequestHandler: 24

Eliminated Races	
E1: Race on field org.apache.ftpserver.RequestHandler. m_isConnectionClosed	
org.apache.ftpserver.RequestHandler: 13	org.apache.ftpserver.RequestHandler: 15

… To This?

Detected Races	
R1: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 9	org.apache.ftpserver.RequestHandler: 18

Eliminated Races	
E1: Race on field org.apache.ftpserver.RequestHandler. m_isConnectionClosed	
org.apache.ftpserver.RequestHandler: 13	org.apache.ftpserver.RequestHandler: 15
E2: Race on field org.apache.ftpserver.RequestHandler.m_request	
org.apache.ftpserver.RequestHandler: 17	org.apache.ftpserver.RequestHandler: 18
E3: Race on field org.apache.ftpserver.RequestHandler.m_writer	
org.apache.ftpserver.RequestHandler: 19	org.apache.ftpserver.RequestHandler: 20
E4: Race on field org.apache.ftpserver.RequestHandler.m_reader	
org.apache.ftpserver.RequestHandler: 21	org.apache.ftpserver.RequestHandler: 22
E5: Race on field org.apache.ftpserver.RequestHandler.m_controlSocket	
org.apache.ftpserver.RequestHandler: 23	org.apache.ftpserver.RequestHandler: 24

An Example: Static Datarace Analysis

Input relations:

next(p1, p2), mayAlias(p1, p2), guarded(p1, p2)

Output relations:

parallel(p1, p2), race(p1, p2)

Constraints:

parallel(p3, p2) :- parallel(p1, p2), next(p3, p1).

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

…

An Example: Static Datarace Analysis

Input relations:

next(p1, p2), mayAlias(p1, p2), guarded(p1, p2)

Output relations:

parallel(p1, p2), race(p1, p2)

Constraints:

parallel(p3, p2) :- parallel(p1, p2), next(p3, p1).

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

…

program point p1 is immediate successor of p2.

p1 & p2 may access the same memory location.

p1 & p2 are guarded by the same lock.

p1 & p2 may happen in parallel.

p1 & p2 may have a datarace.

If p1 & p2 may happen in parallel, and p3 is successor of p1, then p3 & p2 may happen in parallel.

If p2 & p1 may happen in parallel, then p1 & p2 may happen in parallel.

If p1 & p2 may happen in parallel, and they may access the same memory location, and they are not guarded by the same lock, then p1 & p2 may have a datarace.

An Example: Static Datarace Analysis

Input relations:

next(p1, p2), mayAlias(p1, p2), guarded(p1, p2)

Output relations:

parallel(p1, p2), race(p1, p2)

```
a = 1;
if (a > 2) { // p1
  ... // p3
}
```

Constraints:

parallel(p3, p2) :- parallel(p1, p2), next(p3, p1). **weight 5**

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

…

¬race(x2, x1). **weight 25**

"Soft" Rule

"Hard" Rule

An Example: Static Datarace Analysis

```
public class RequestHandler {
    Request request;
    FtpWriter writer;
    BufferedReader reader;
    Socket controlSocket;
    boolean isConnectionClosed;
    …

  public Request getRequest() {
        return request;
  }
```

```
public void close() {
    synchronized (this) {
        if (isClosed)
            return;
        isClosed = true;
    }
    request.clear();
    request = null;
    writer.close();
    writer = null;
    reader.close();
    reader = null;
    controlSocket.close();
    controlSocket = null;
}
```

Source code snippet from Apache FTP Server

An Example: Static Datarace Analysis

```
public class RequestHandler {
    Request request;
    FtpWriter writer;
    BufferedReader reader;
    Socket controlSocket;
    boolean isConnectionClosed;
    …

 public Request getRequest() {
        return request;
 }
```

R1

```
public void close() {
    synchronized (this) {
        if (isClosed)
            return;
        isClosed = true;
    }
    request.clear();
    request = null;
    writer.close();
    writer = null;
    reader.close();
    reader = null;
    controlSocket.close();
    controlSocket = null;
}
```

Source code snippet from Apache FTP Server

An Example: Static Datarace Analysis

```
public class RequestHandler {
    Request request;
    FtpWriter writer;
    BufferedReader reader;
    Socket controlSocket;
    boolean isConnectionClosed;
    …

 public Request getRequest() {
        return request;
 }
```

```
public void close() {
    synchronized (this) {
      if (isClosed)
          return;
      isClosed = true;
    }
    request.clear();
    request = null;
    writer.close();
    writer = null;
    reader.close();
    reader = null;
    controlSocket.close();
    controlSocket = null;
}
```

R2 R3 R4 R5

Source code snippet from Apache FTP Server

How Does Generalization Work?

```
...
request.clear();// x1
request = null; // x2
writer.close(); // y1
writer = null;  // y2
...
```

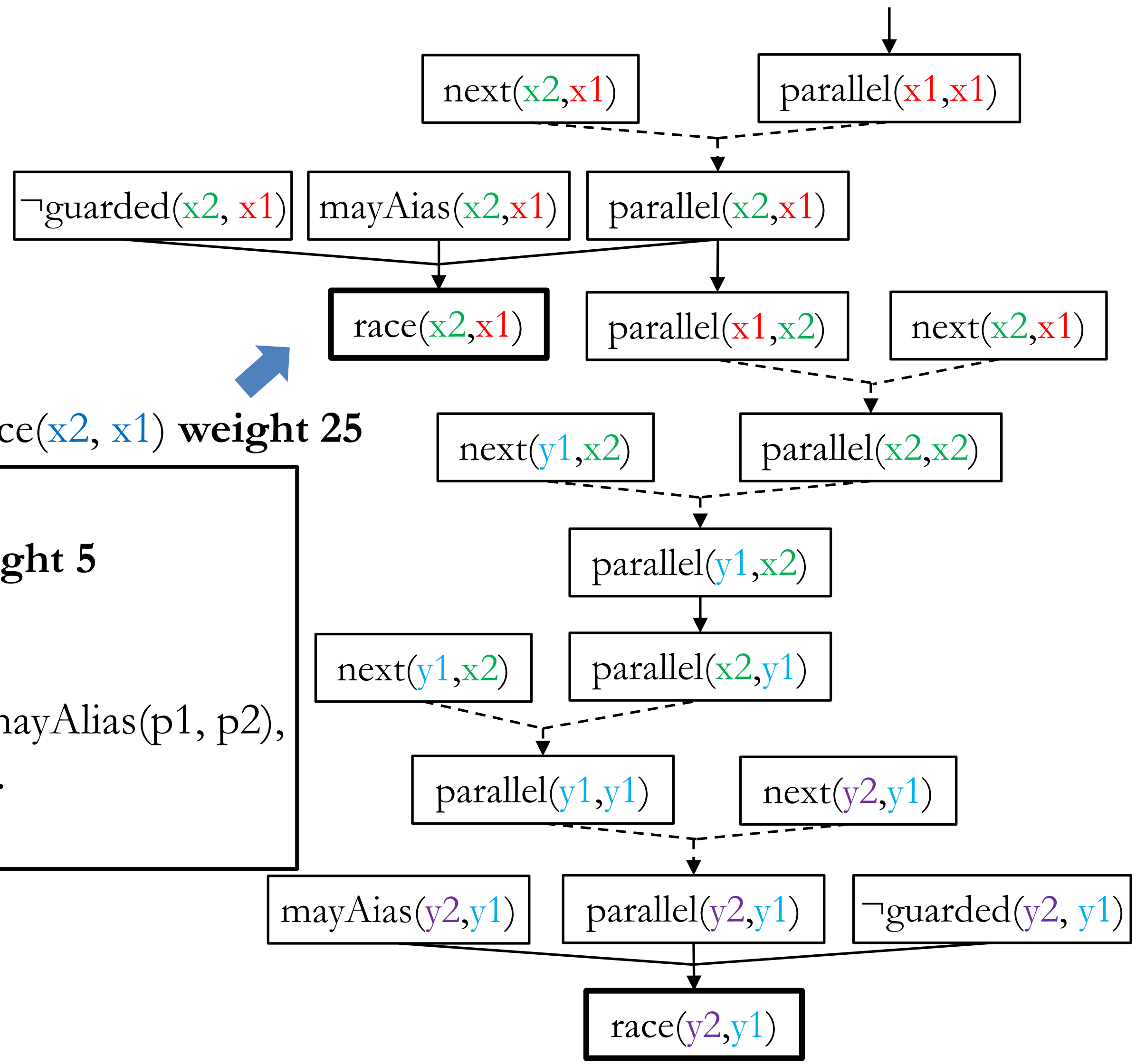
¬race(x2, x1) **weight 25**

parallel(p3, p2) :- parallel(p1, p2),
next (p3, p1). **weight 5**

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2),
¬guarded(p1, p2).

...



How Does Generalization Work?

```
        ...
    request.clear();// x1
    request = null; // x2
    writer.close(); // y1
    writer = null;  // y2
        ...
```

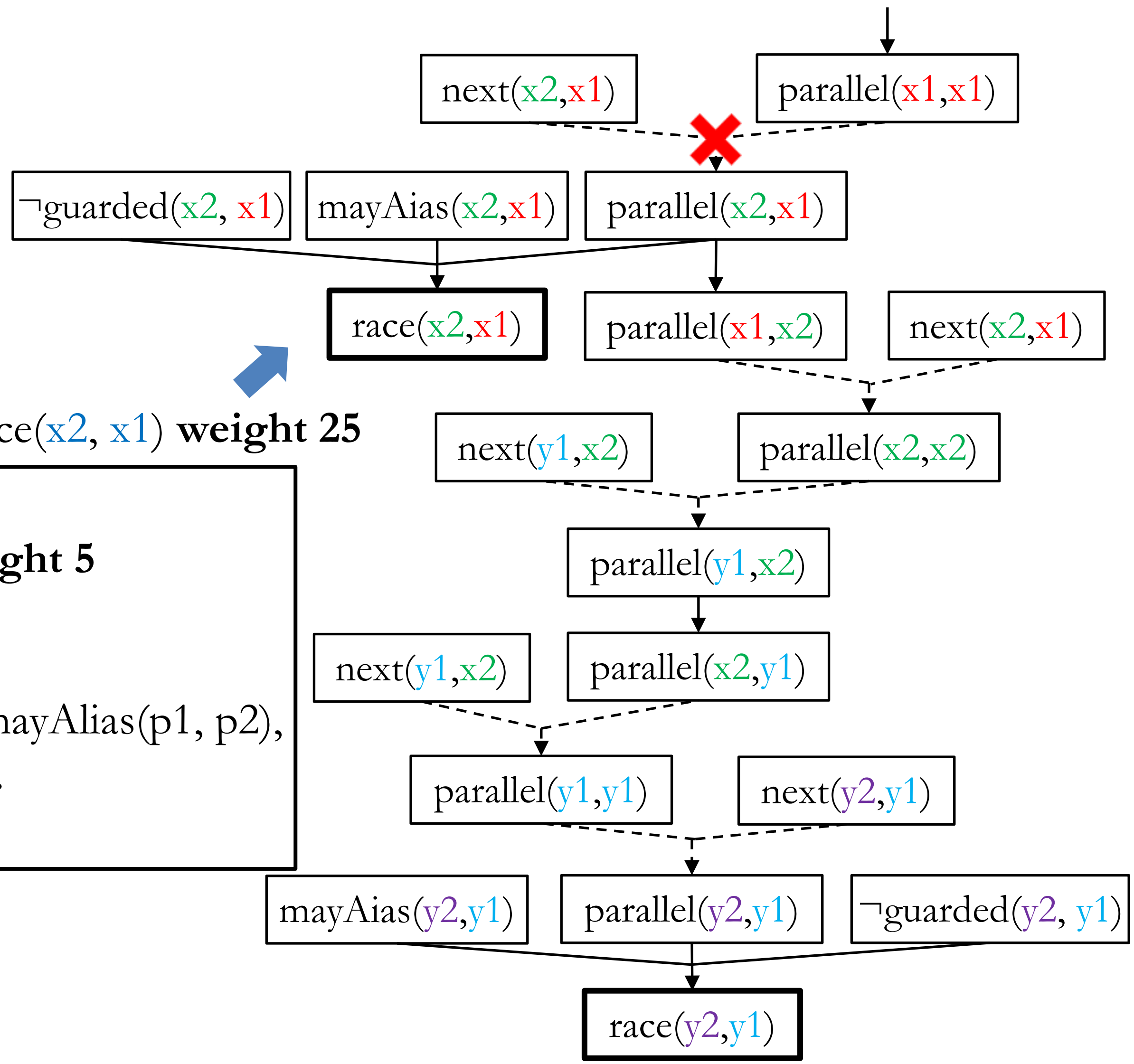
¬race(x2, x1) **weight 25**

parallel(p3, p2) :- parallel(p1, p2),
next (p3, p1). **weight 5**

parallel(p1, p2) :- parallel(p2, p1).

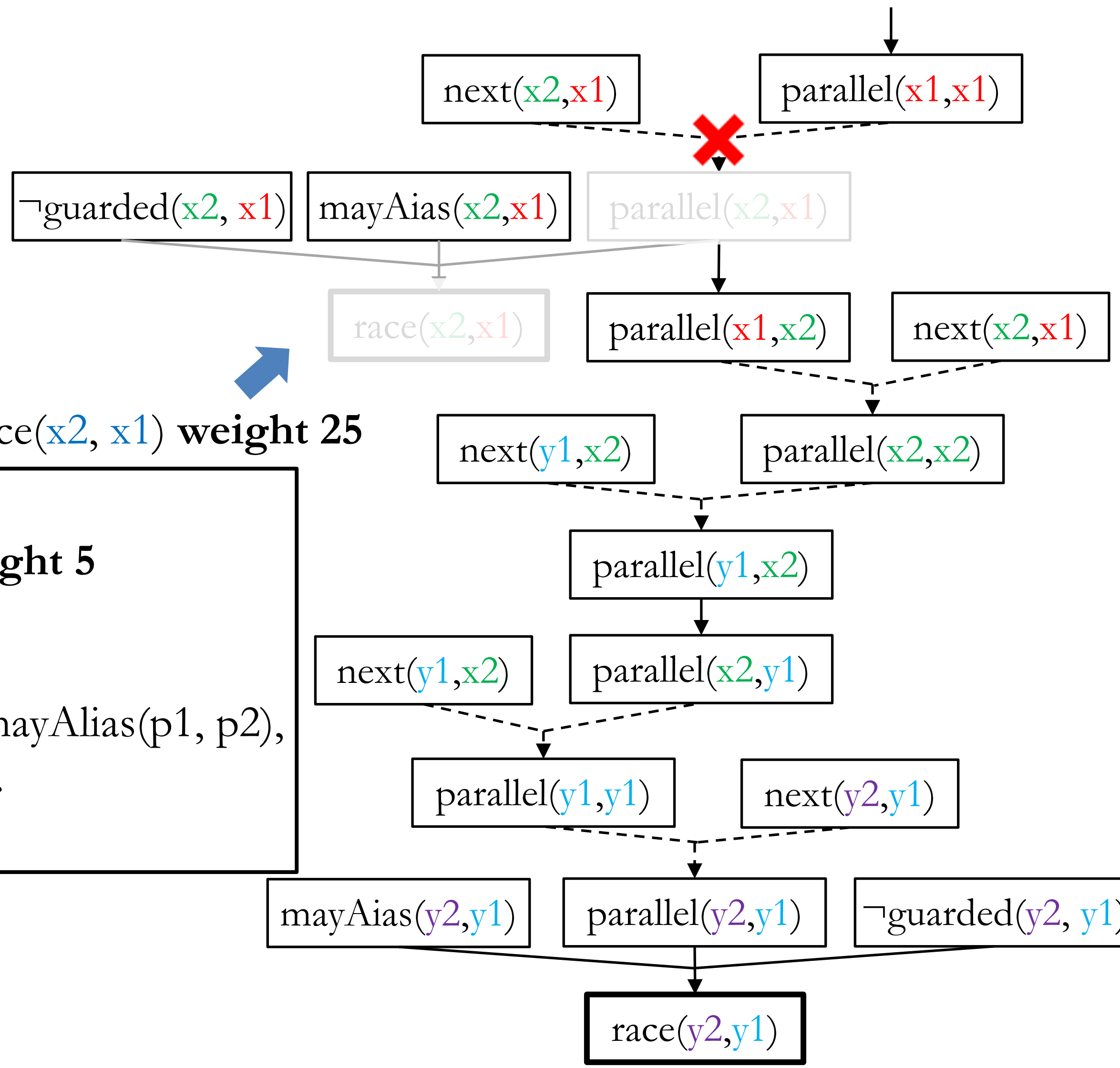
race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2),
¬guarded(p1, p2).

…



How Does Generalization Work?

```
        ...
    request.clear();// x1
    request = null; // x2
    writer.close(); // y1
    writer = null;  // y2
        ...
```



¬race(x2, x1) **weight 25**

parallel(p3, p2) :- parallel(p1, p2),
next (p3, p1). **weight 5**

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2),
¬guarded(p1, p2).

…

How Does Generalization Work?

```
...
request.clear();// x1
request = null; // x2
writer.close(); // y1
writer = null;  // y2
...
```

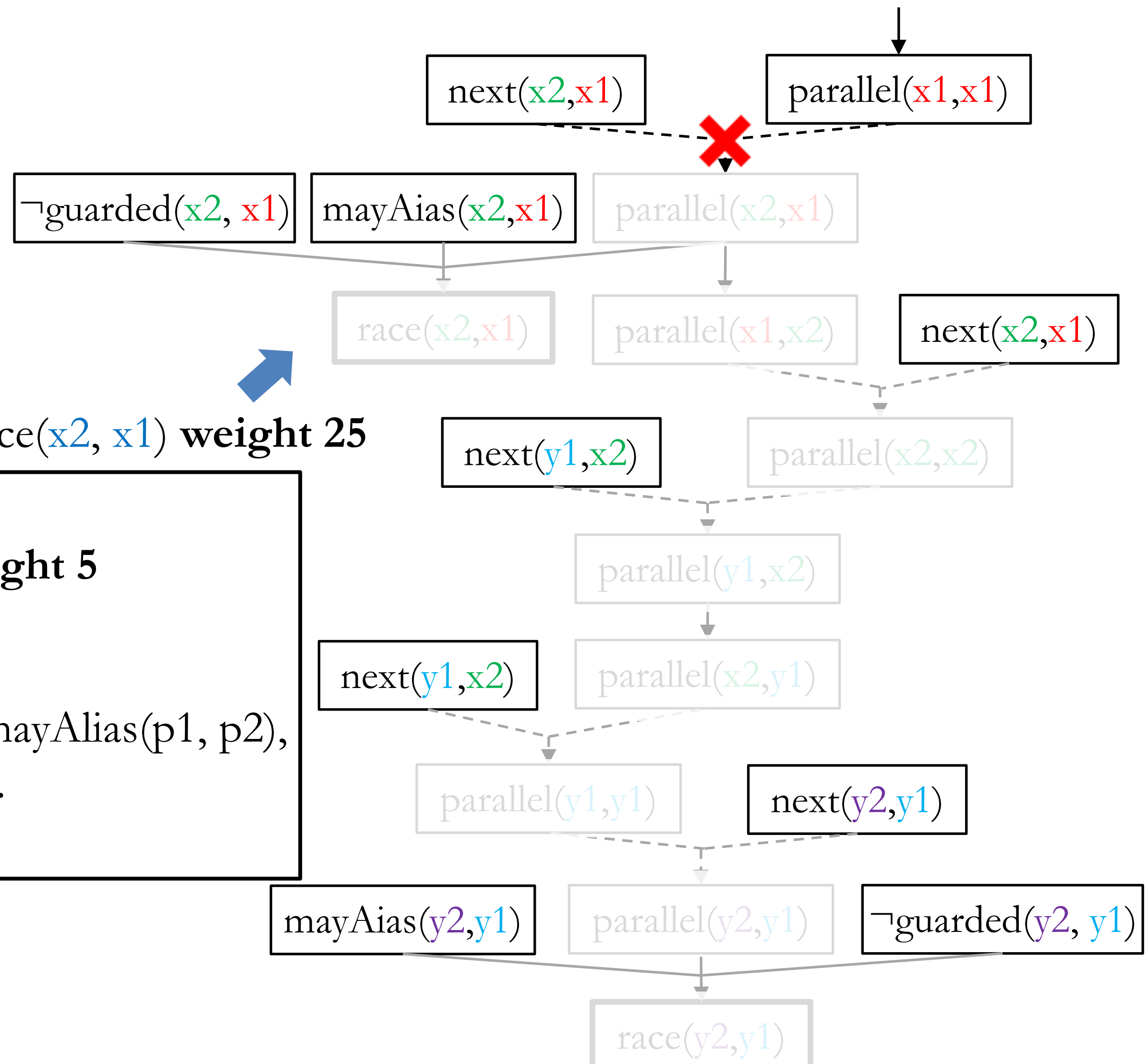
¬race(x2, x1) **weight 25**

parallel(p3, p2) :- parallel(p1, p2),
next (p3, p1). **weight 5**

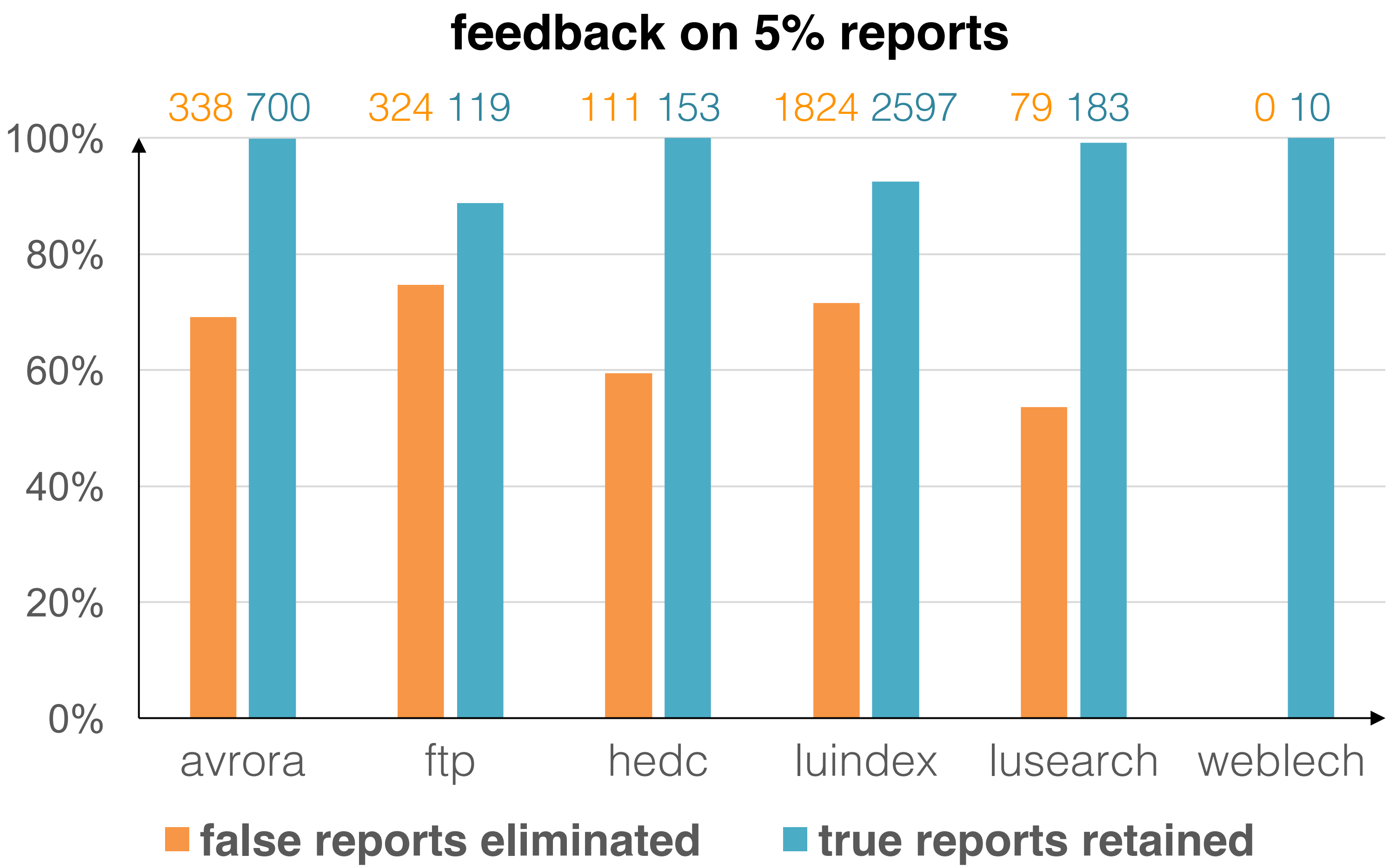
parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2),
¬guarded(p1, p2).

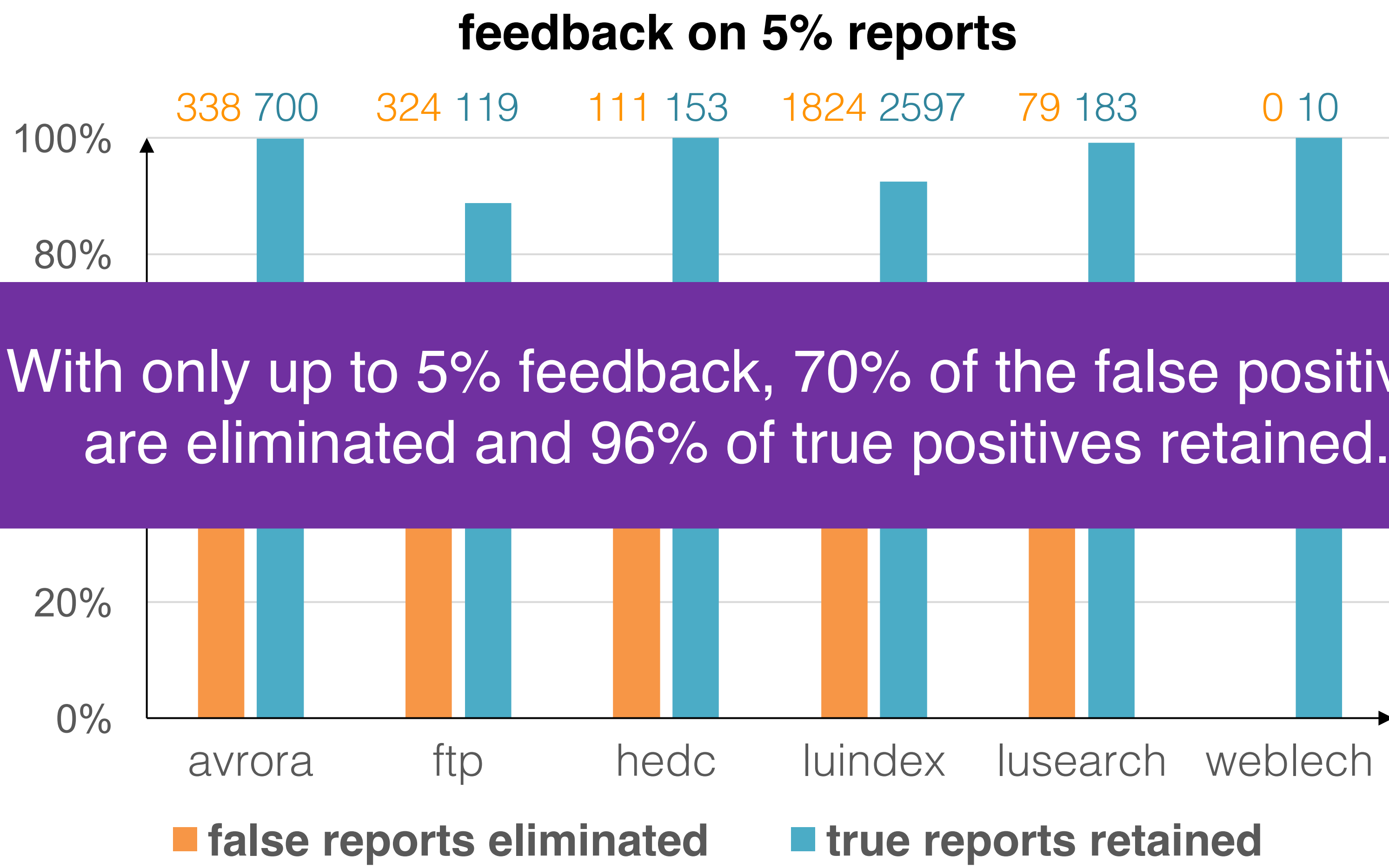
...



Accuracy of Alarm Classification

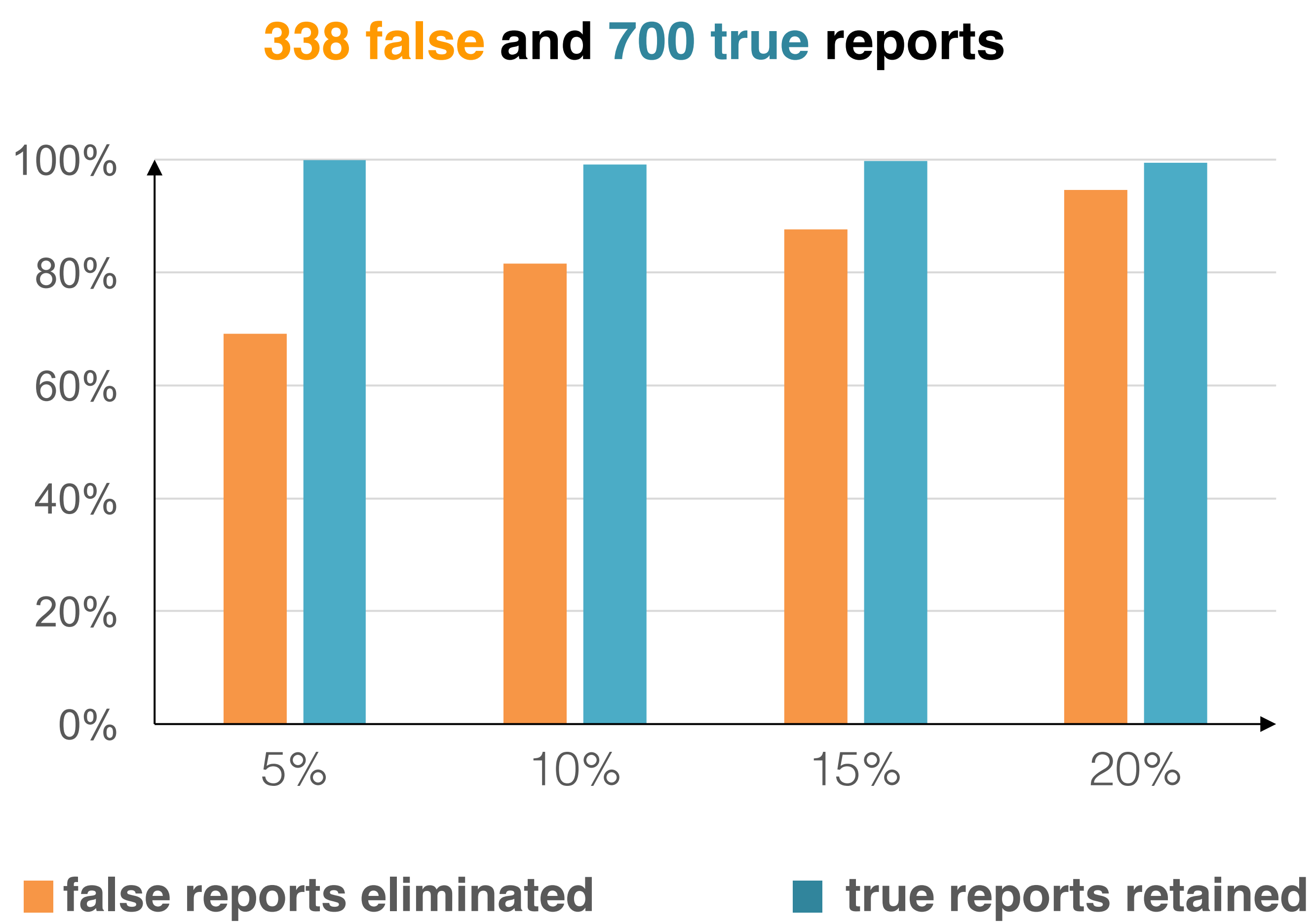


Accuracy of Alarm Classification



With only up to 5% feedback, 70% of the false positives are eliminated and 96% of true positives retained.

Impact of Varying Amount of Feedback



Overview of Applications

- Abstraction Selection [PLDI 2014]
- Alarm Classification [FSE 2015]
- Alarm Resolution [OOSPLA 2017]

Example Revisited: Static Datarace Analysis

Can this statement be executed by different threads in parallel?

```java
public Request getRequest() {
        return request;
}
```

```java
public void close() {
    synchronized (this) {
        if (isClosed)
            return;
        isClosed = true;
    }
    request.clear();
    request = null;
    writer.close();
    writer = null;
    reader.close();
    reader = null;
    controlSocket.close();
    controlSocket = null;
}
```

Source code snippet from Apache FTP Server

Illustration: Space of Questions

```
...
request.clear();// x1
request = null; // x2
writer.close(); // y1
writer = null;  // y2
...
```

parallel(p3, p2) :- parallel(p1, p2), next (p3, p1).

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

...

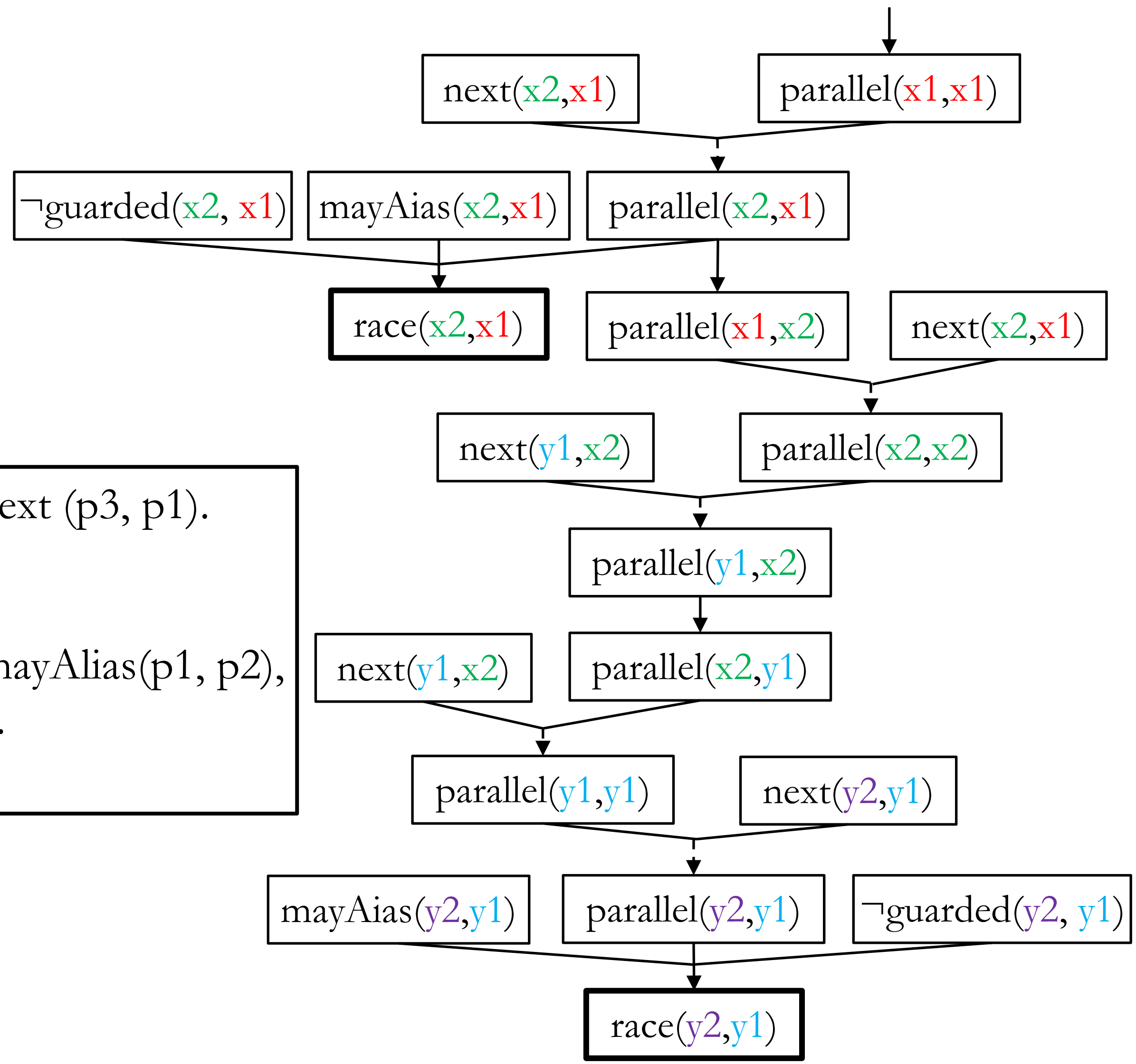


Illustration: Space of Questions

```
        ...
request.clear();// x1
request = null; // x2
writer.close(); // y1
writer = null;  // y2
        ...
```

parallel(p3, p2) :- parallel(p1, p2), next (p3, p1).

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

…

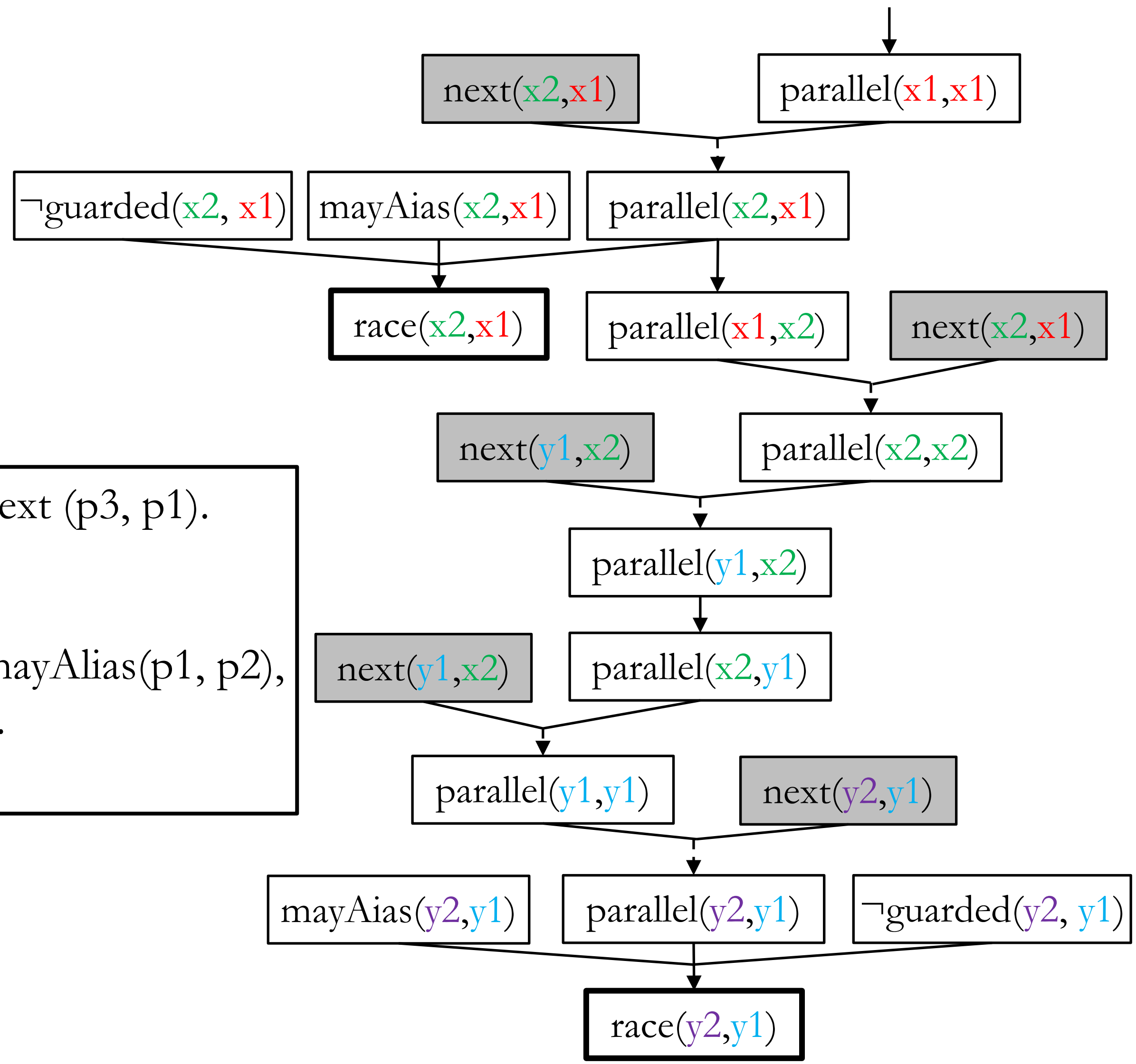


Illustration: Space of Questions

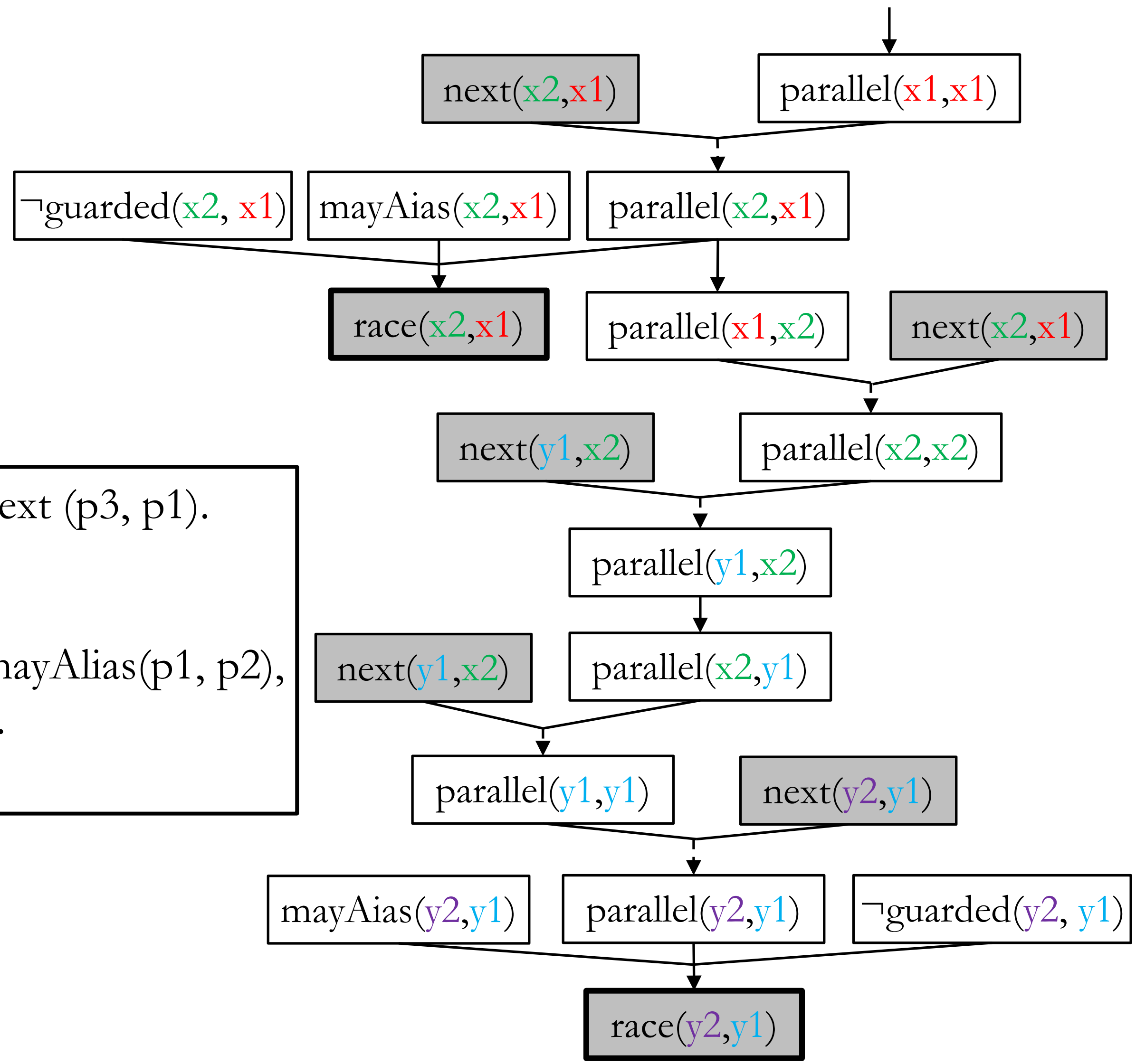
```
        ...
request.clear();// x1
request = null; // x2
writer.close(); // y1
writer = null;  // y2
        ...
```

parallel(p3, p2) :- parallel(p1, p2), next (p3, p1).

parallel(p1, p2) :- parallel(p2, p1).

race(p1, p2) :- parallel(p1, p2), mayAlias(p1, p2), ¬guarded(p1, p2).

…



Two Key Objectives

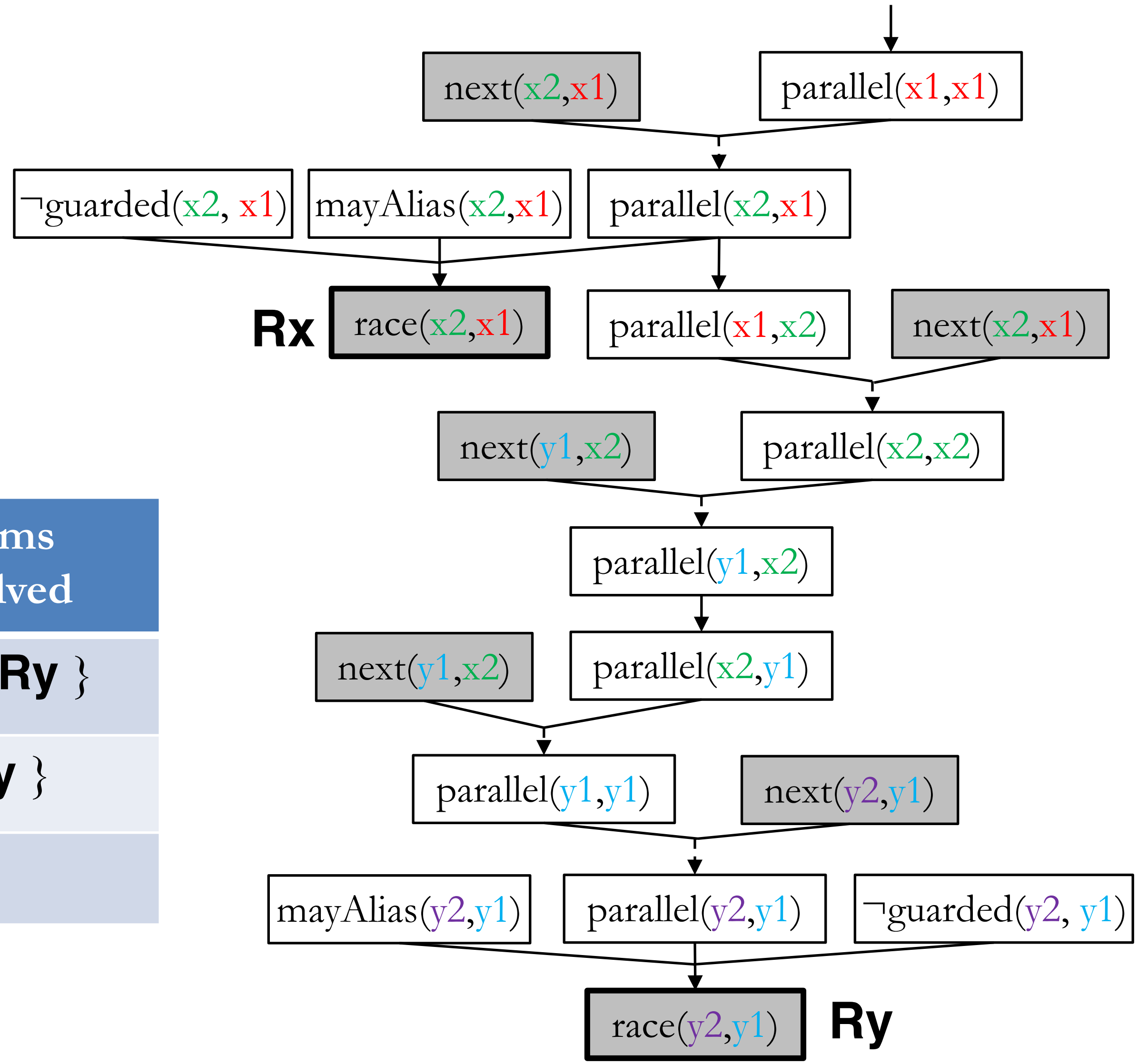
- **Generalization**
  - Number of Questions << Number of Alarms Resolved

- **Prioritization**
  - Prioritize Questions Likely to Resolve the Most Alarms

Illustration: Payoff Comparison

```
        ...
    request.clear();// x1
    request = null; // x2
    writer.close(); // y1
    writer = null;  // y2
        ...
```

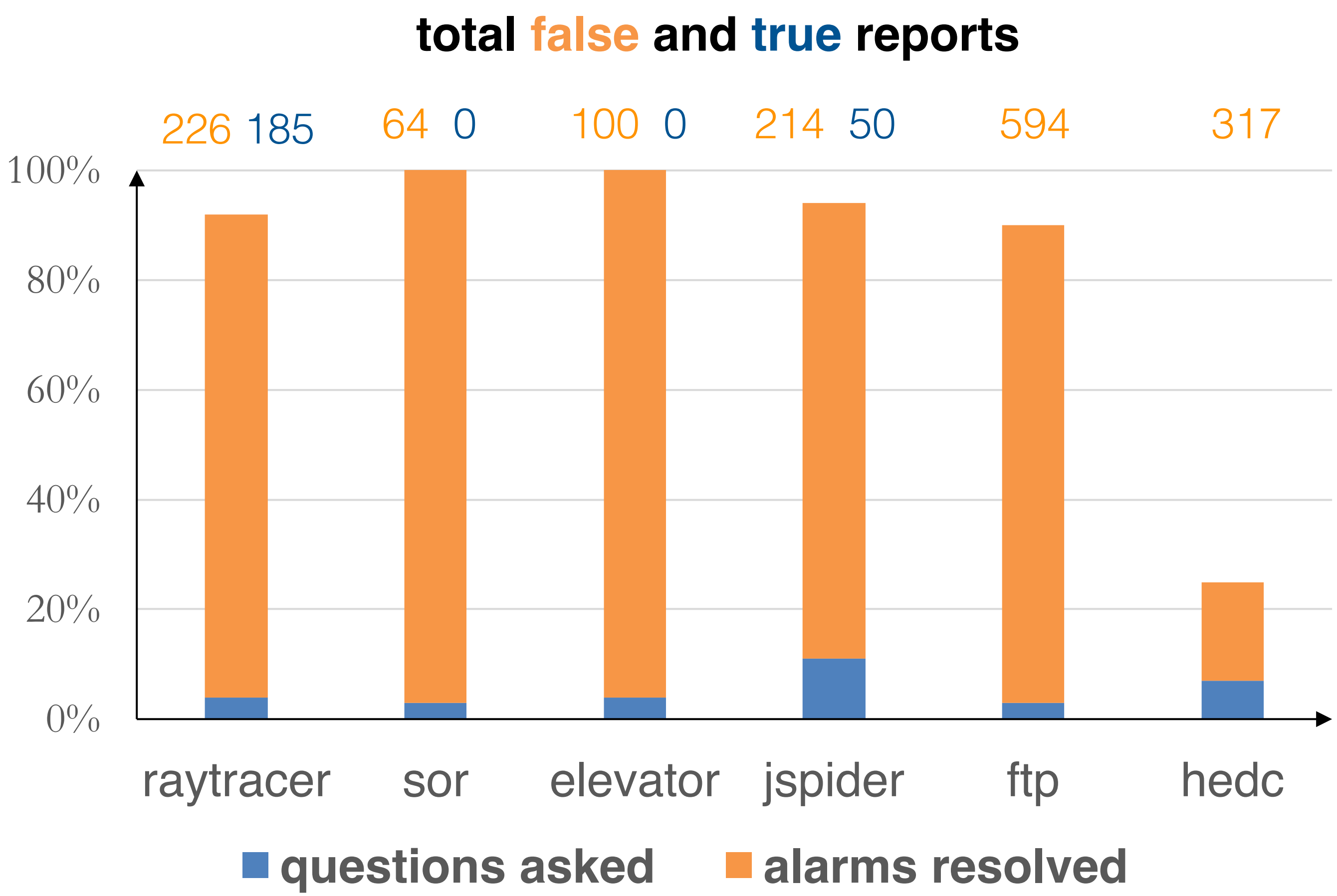
Questions Asked	Alarms Resolved
{ parallel(x1, x1) }	{ **Rx**, **Ry** }
{ parallel(y1, y1) }	{ **Ry** }
{ mayAlias(y2, y1) }	Ø



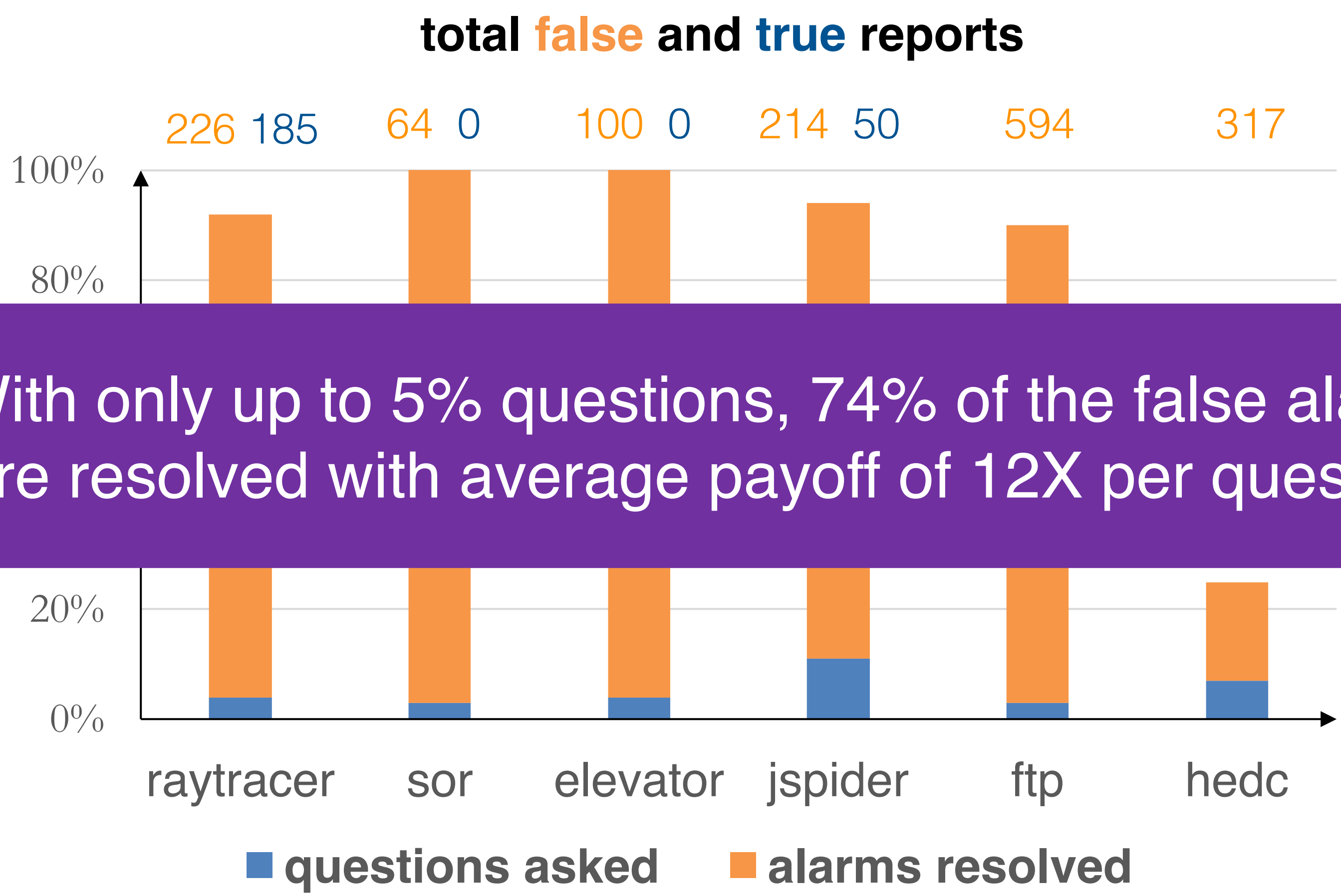
Highlights of Overall Approach

- Iterative: leverages labels of past questions in choosing future questions
- Maximizes the expected payoff in each iteration
  - Payoff = # Alarms Resolved / # Questions Asked
- Non-linear optimization objective
  - Binary search on payoff by solving sequence of MaxSAT instances
- Data-driven: leverages heuristics to guess likely labels
  - Static, Dynamic, Aggregated

Empirical Results: Generalization

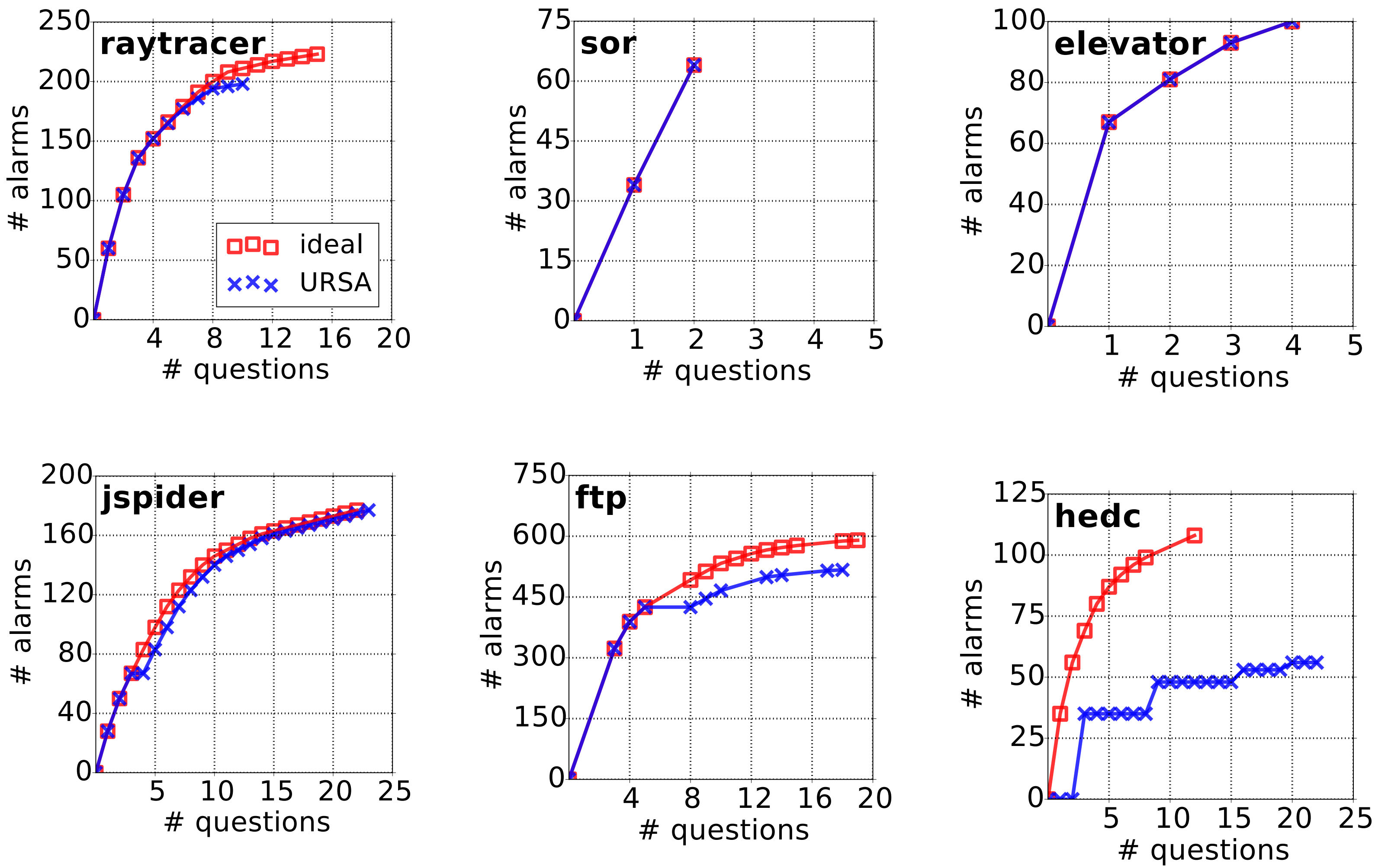


Empirical Results: Generalization

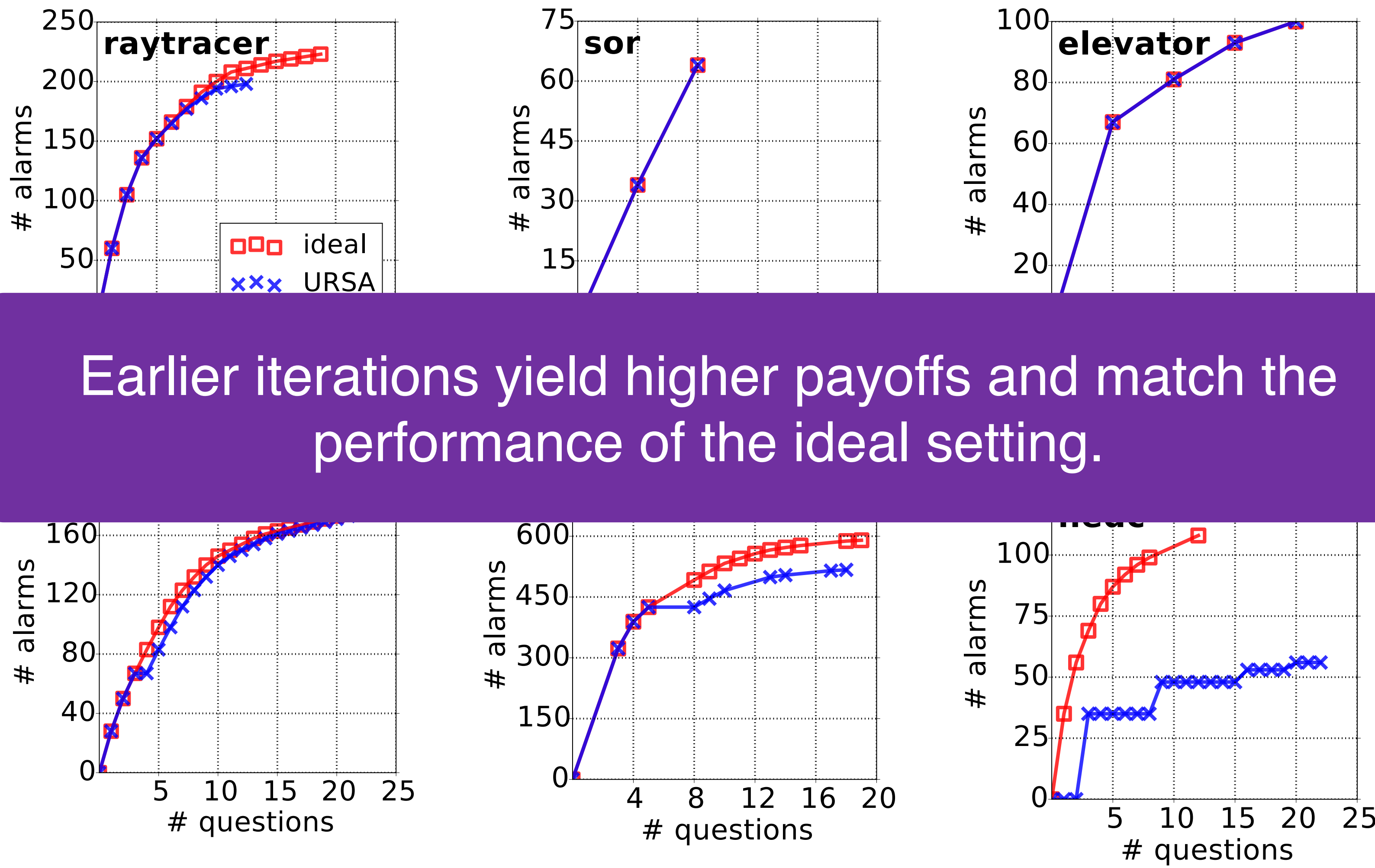


With only up to 5% questions, 74% of the false alarms are resolved with average payoff of 12X per question.

Empirical Results: Prioritization



Empirical Results: Prioritization



Earlier iterations yield higher payoffs and match the performance of the ideal setting.

Summary: Applications in Software Analysis

	Hard Constraints	Soft Constraints
Automated Verification [PLDI'14]	analysis rules $\text{abstraction}_1 \oplus \ldots \oplus \text{abstraction}_n$	¬ result_i **weight w_i** **query resolution award** abstraction_j **weight w_j** **abstraction cost**
Static Bug Detection [FSE'15]	analysis rules	analysis rule_i **weight w_i** **confidence of writer** ¬ result_j **weight w_j** **confidence of user**
Interactive Verification [OOPSLA'17]	analysis rules	¬ cause_i **weight w_i** **cost of inspection** result_j **weight w_j** **reward of resolution**

Other Applications

- Statistical Relational Learning

```
wrote(p, t) :- advisedBy(s, p), wrote(s, t). weight 3
p == q :- advisedBy(s, p), advisedBy(s, q).  weight 5
professor(p) :- advisedBy(_, p).             weight 20
wrote("Tom", "paper1").
wrote("Tom", "paper2").
wrote("Jerry", "paper1").
wrote("Chuck", "paper2").
professor("Jerry").
```

Given constraints and facts, find most likely answer to:

`advisedBy("Tom", ?)`

- Mathematical Programming

```
totalShelf [] += stock[p] * space [p]
totalProfit[] += stock[p] * profit[p]
product(p) -> stock[p]     >= minStock[p]
product(p) -> stock[p]     <= maxStock[p]
true       -> totalShelf[] <= maxShelf[]
lang:solve:variable(`stock)
lang:solve:max(`totalProfit)
```

Talk Outline

- Background
- Part I: Applications in Software Analysis
- Part II: Techniques for MaxSAT Solving
- Conclusion

The Inference Problem

Input relations:
edge(x, y)
Output relations:
path(x, y)
Hard constraints:
path(x, x).
path(x, z) :- path(x, y), edge(y, z).
Soft constraints:
¬ path(x, y). **weight 1.5**

Markov Logic Network

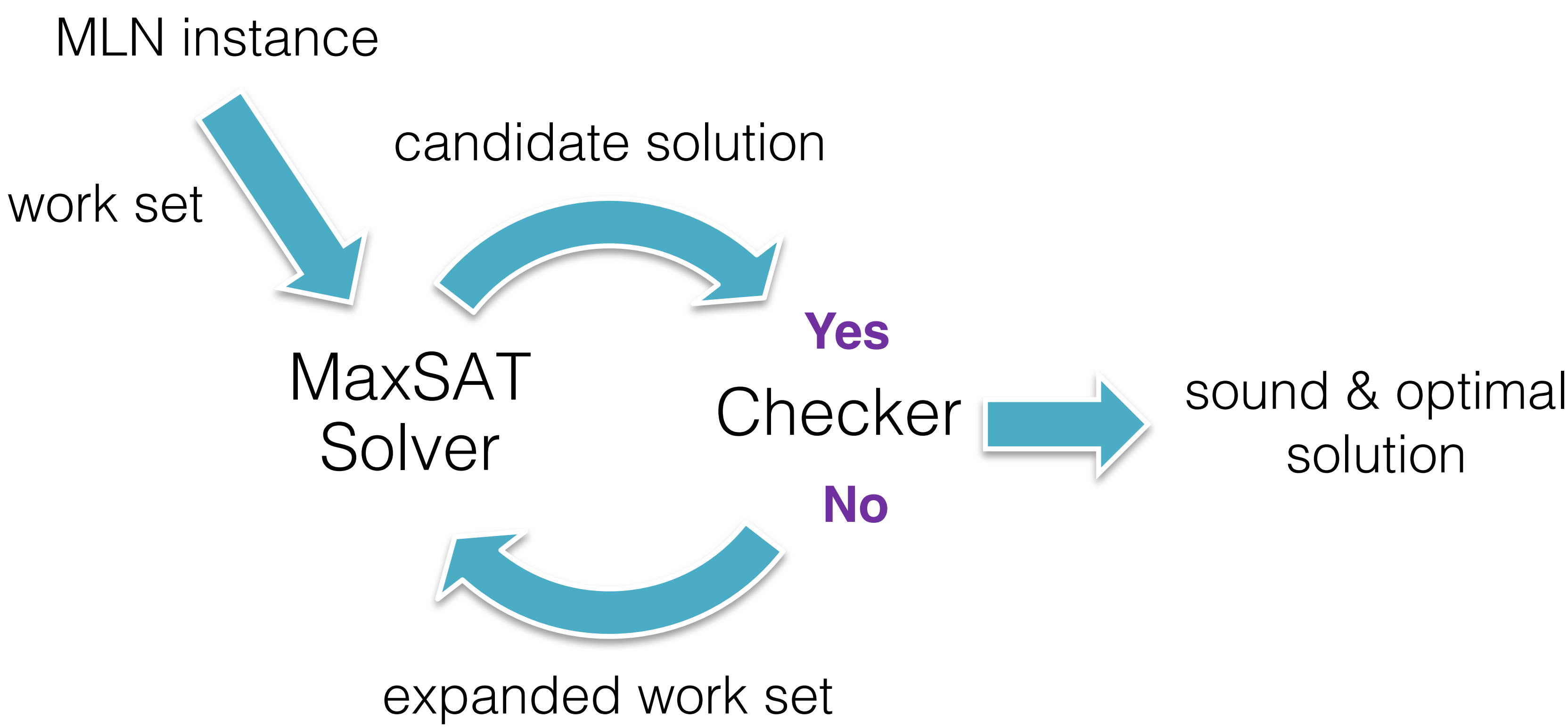
Solver

Existing Solvers	**Soundness**	**Optimality**	**Scalability**
Tuffy [VLDB'11]	✗	✗	✓
Alchemy [ML'06]	✗	✗	✓
CPI [UAI'08]	✓	✓	✗
RockIt [AAAI'13]	✓	✓	✗
Z3 [TACAS'08]	✓	✓	✗

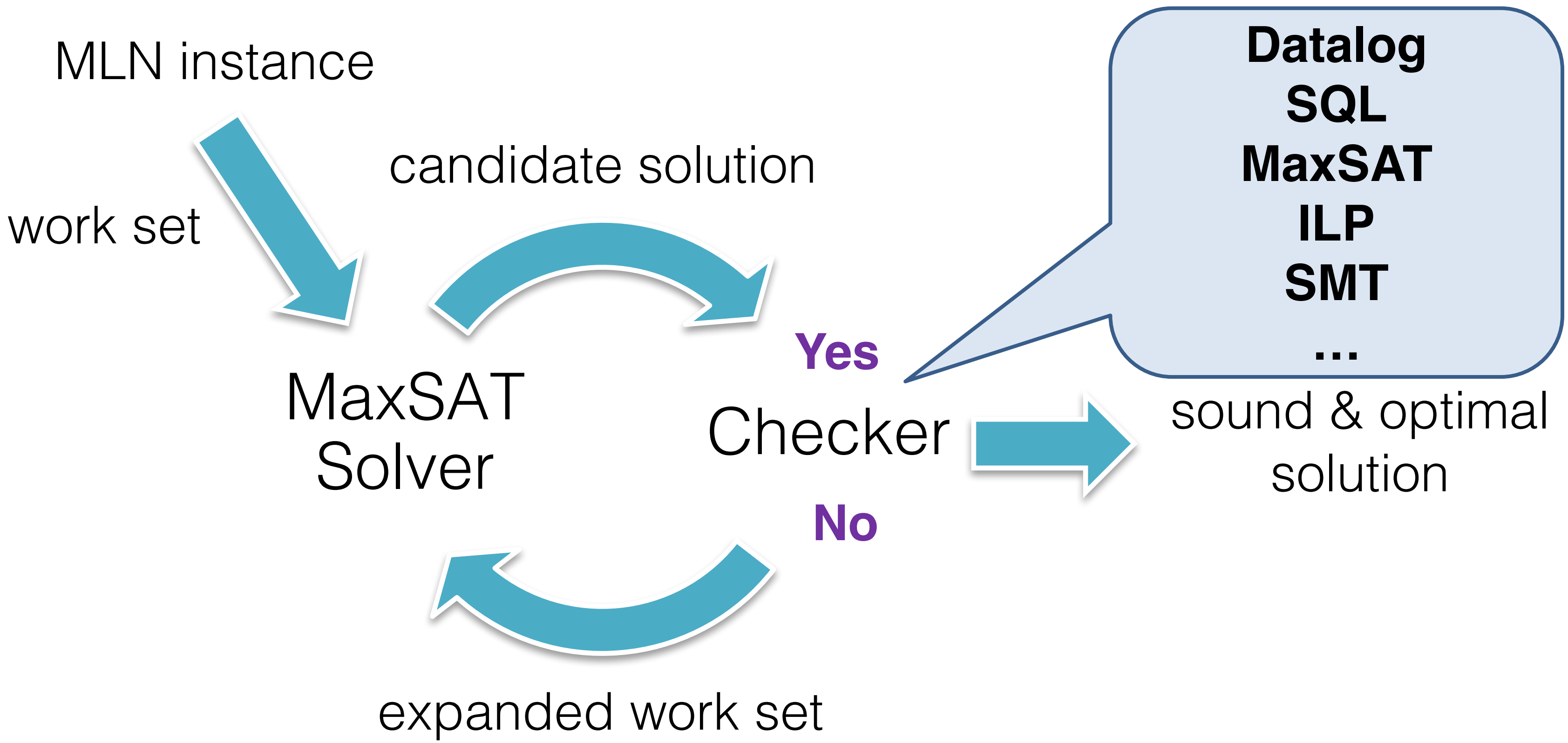
Overview of Techniques

- General Framework [SAT 2015]
- Bottom-Up Solving [AAAI 2016]
- Top-Down Solving [POPL 2016]

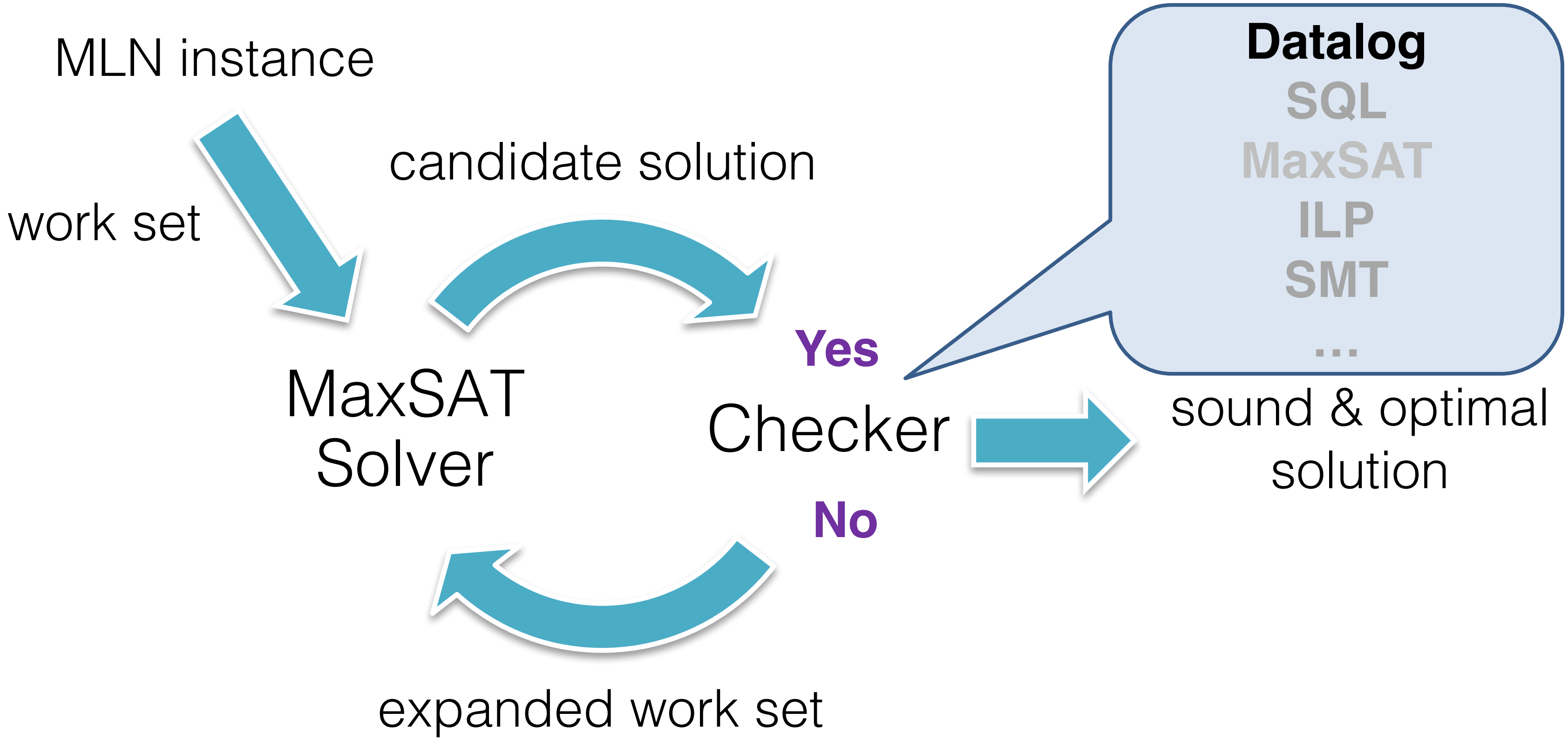
Framework Architecture



Framework Instances

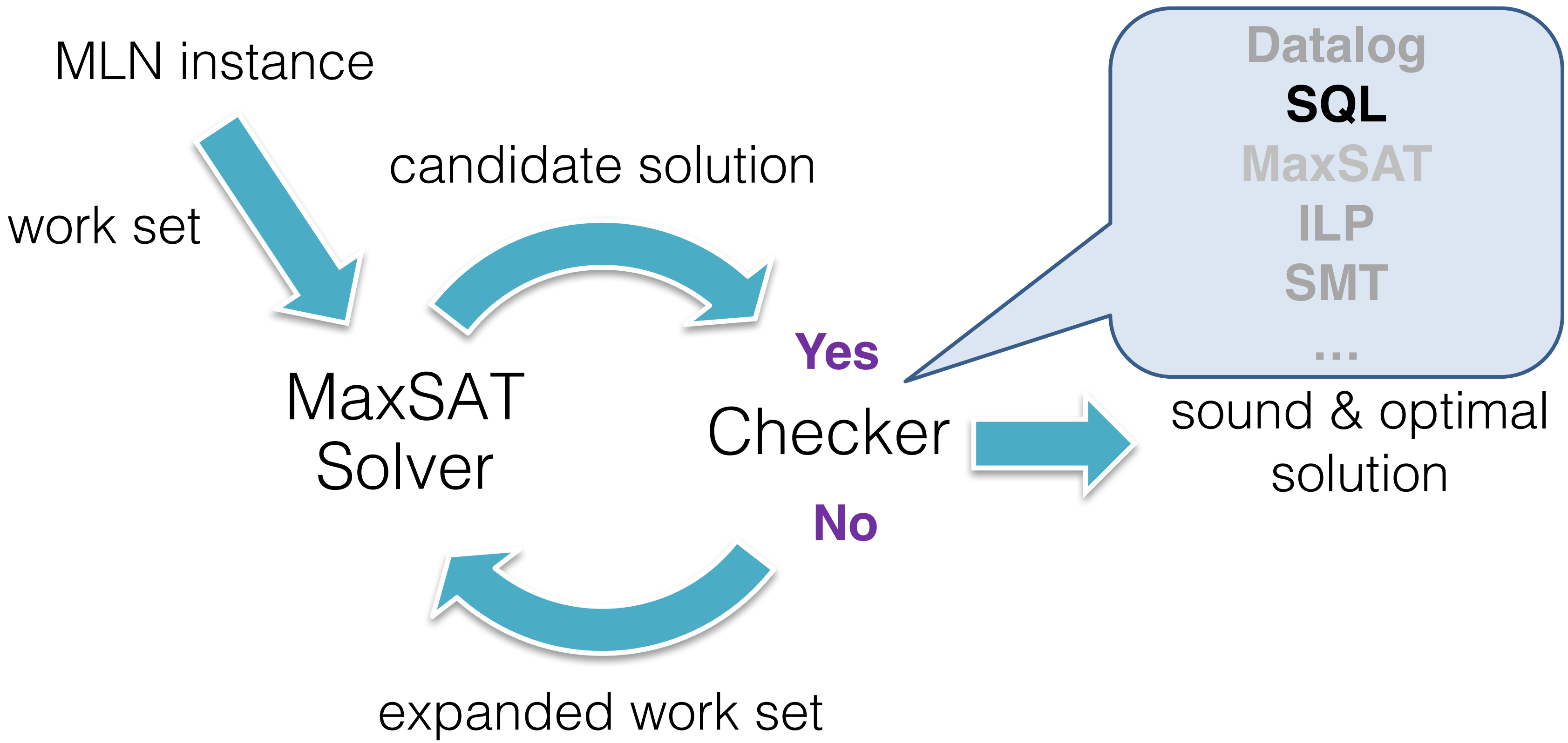


Framework Instance: Abstraction Refinement



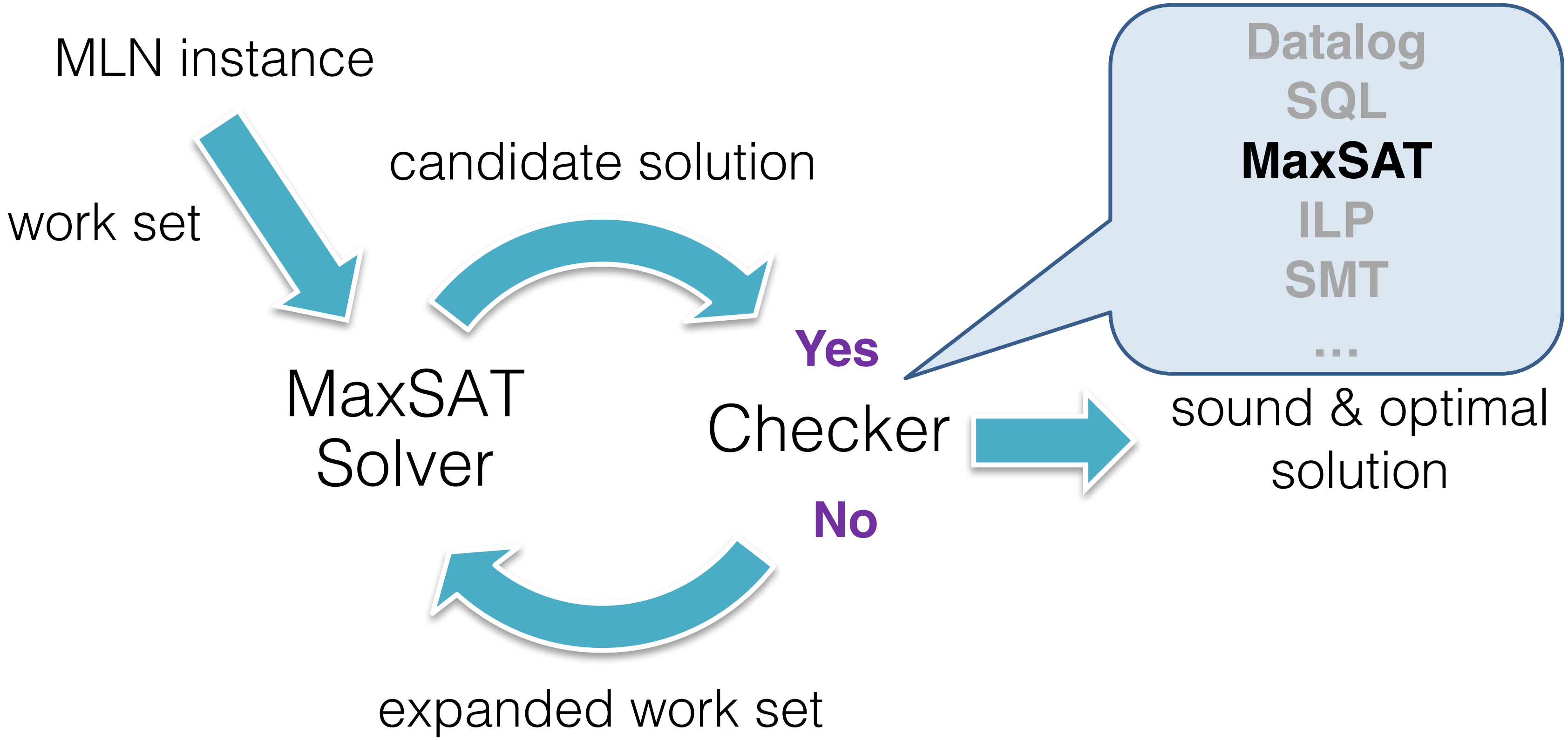
On Abstraction Refinement for Program Analyses in Datalog [PLDI 2014]

Framework Instance: Bottom-Up Solving



Scaling Relational Inference Using Proofs and Refutations
[AAAI 2016]

Framework Instance: Top-Down Solving



Query-Guided Maximum Satisfiability
[POPL 2016]

Overview of Techniques

- General Framework [SAT 2015]
- Bottom-Up Solving [AAAI 2016]
- Top-Down Solving [POPL 2016]

Bottom-Up Solving

Follows **cutting-plane method [Riedl'09]** with three new insights for better scalability on our applications:

1. Exploits **high-level structure of MLN** to efficiently find new ground constraints violated current solution.

2. Accelerates convergence by eagerly grounding **Horn constraints** using Datalog solver.

3. **Terminates earlier** by checking objective value (rather than set of violated soft constraints) for saturation.

Example

Input relations:

edge(x, y)

Output relations:

path(x, y)

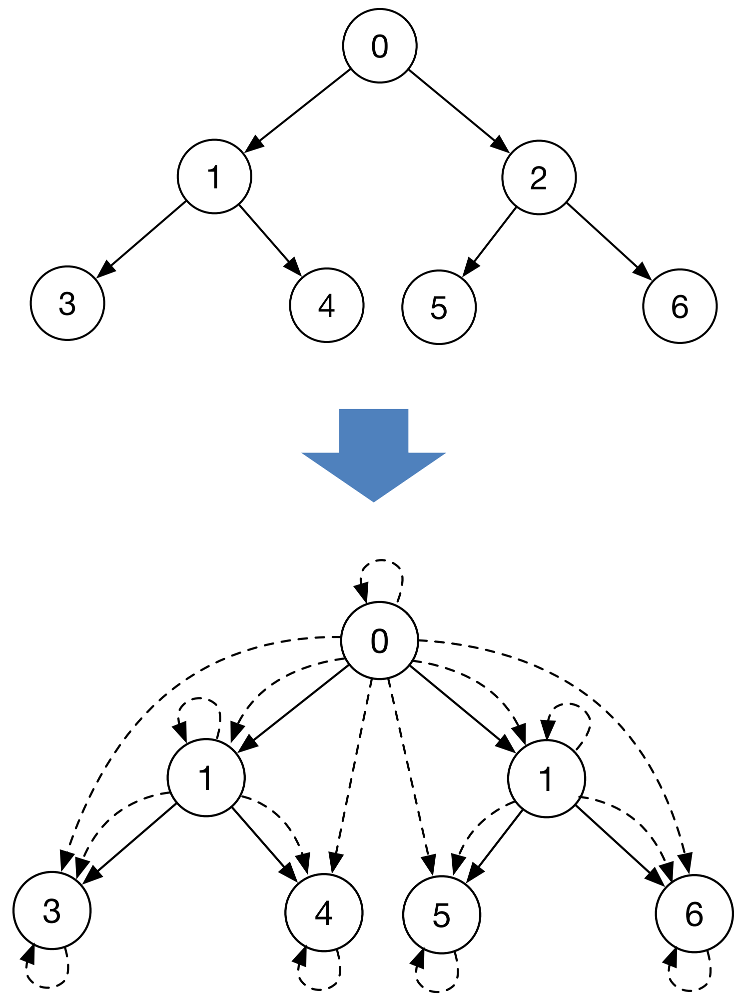
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

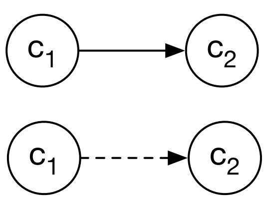
Soft constraints:

¬ path(x, y). **weight 1.5**



edge(c_1,c_2)

path(c_1,c_2)



Example: Iteration 1 - Solve

Input relations:

edge(x, y)

Output relations:

path(x, y)

Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

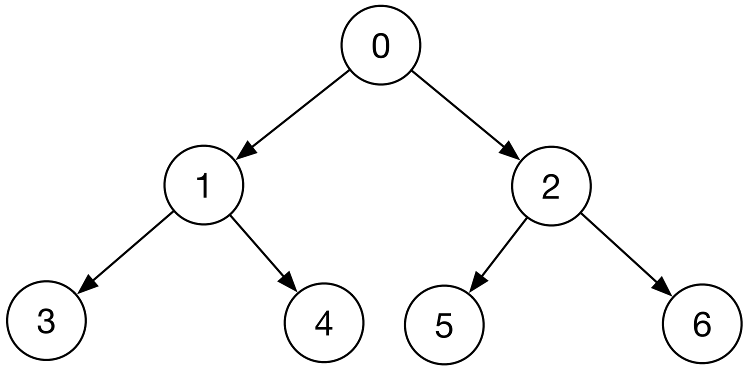
Soft constraints:

¬ path(x, y). **weight 1.5**

Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6)$$

Soft clauses:



Example: Iteration 1 - Check

Input relations:

edge(x, y)

Output relations:

path(x, y)

Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

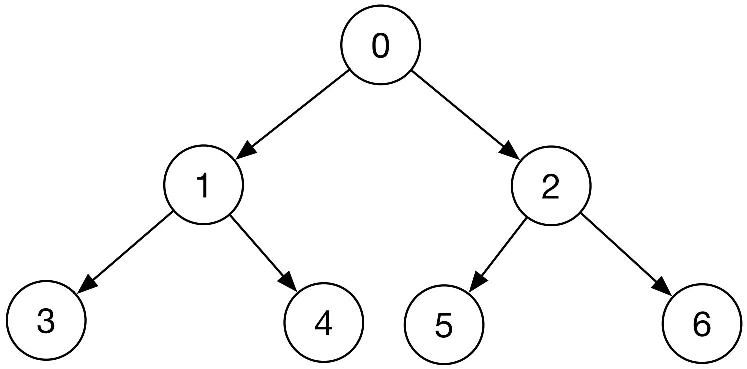
Soft constraints:

¬ path(x, y). **weight 1.5**

Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6)$$

Soft clauses:



Example: Iteration 2 - Solve

Input relations:

edge(x, y)

Output relations:

path(x, y)

Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

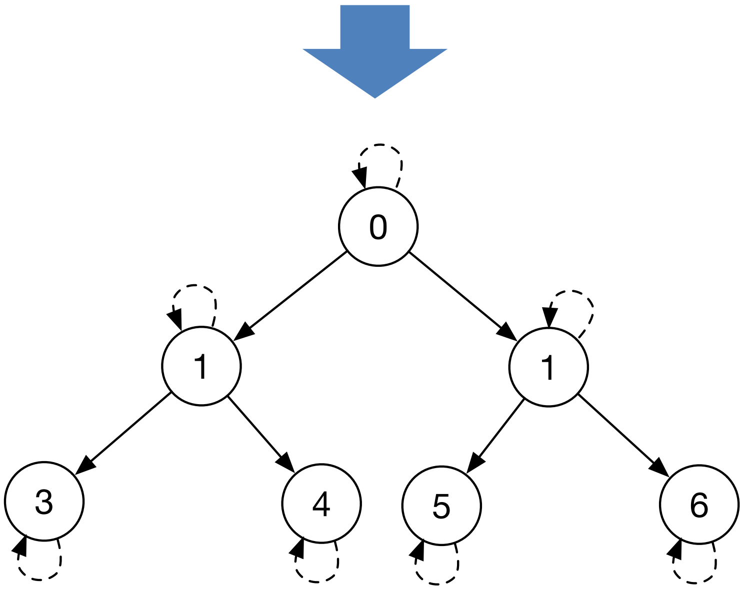
¬ path(x, y). **weight 1.5**

Hard clauses:

$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$

$path(0,0) \wedge \ldots \wedge path(6,6)$

Soft clauses:



Example: Iteration 2 - Check

Input relations:

edge(x, y)

Output relations:

path(x, y)

Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

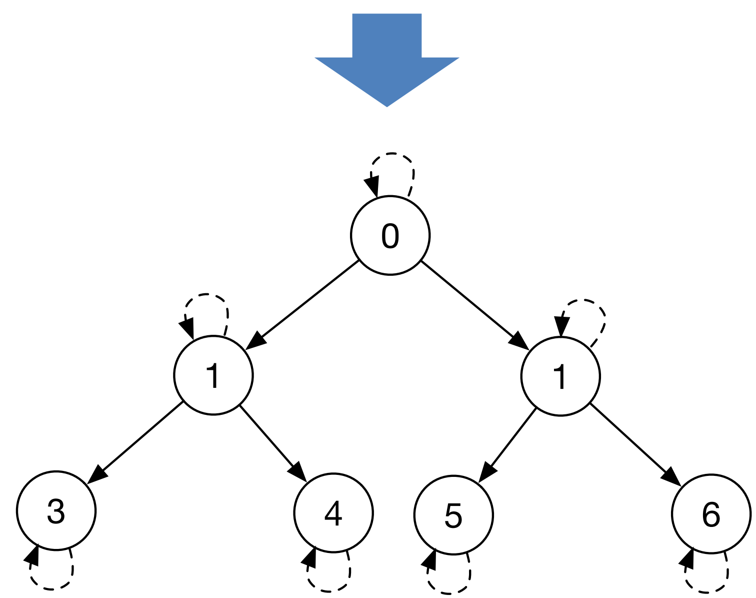
Soft constraints:

¬ path(x, y). **weight 1.5**

Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$$
$$path(0,0) \wedge \ldots \wedge path(6,6)$$

Soft clauses:



Example: Iteration 3 - Solve

Input relations:

edge(x, y)

Output relations:

path(x, y)

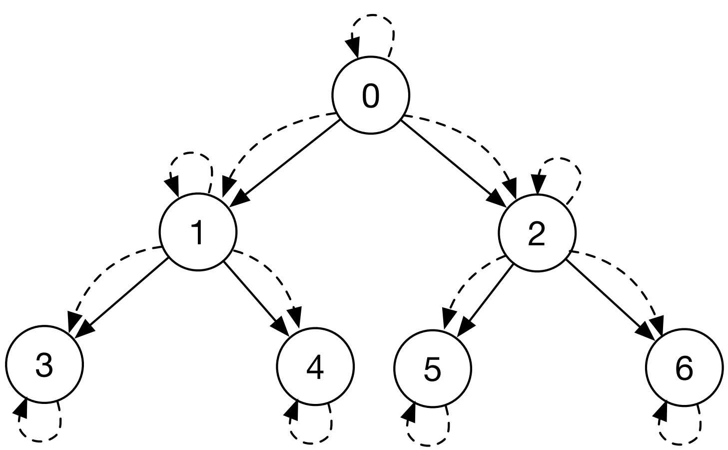
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$$
$$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$$
$$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$$
$$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$$
$$\ldots \wedge$$
$$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6)$$

Soft clauses:

$(\neg path(0,0)$ **weight 1.5**$) \wedge$

$\ldots \wedge$

$(\neg path(6,6)$ **weight 1.5**$)$

Example: Iteration 3 - Check

Input relations:

edge(x, y)

Output relations:

path(x, y)

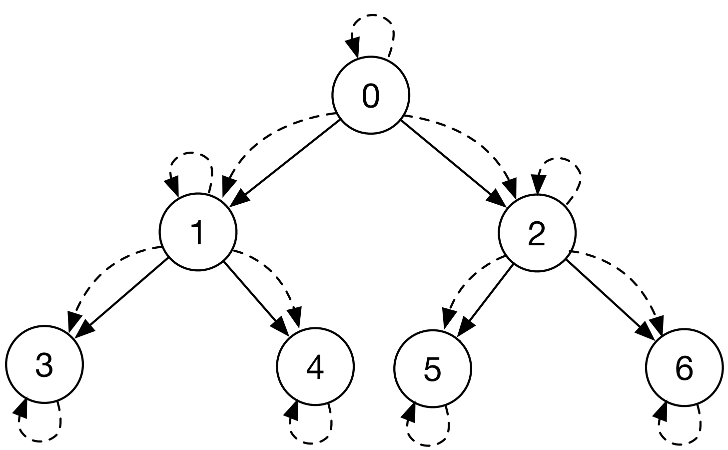
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$$
$$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$$
$$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$$
$$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$$
$$\ldots \wedge$$
$$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6)$$

Soft clauses:

$(\neg path(0,0)$ **weight 1.5**$) \wedge$

$\ldots \wedge$

$(\neg path(6,6)$ **weight 1.5**$)$

Example: Iteration 4 - Solve

Input relations:

edge(x, y)

Output relations:

path(x, y)

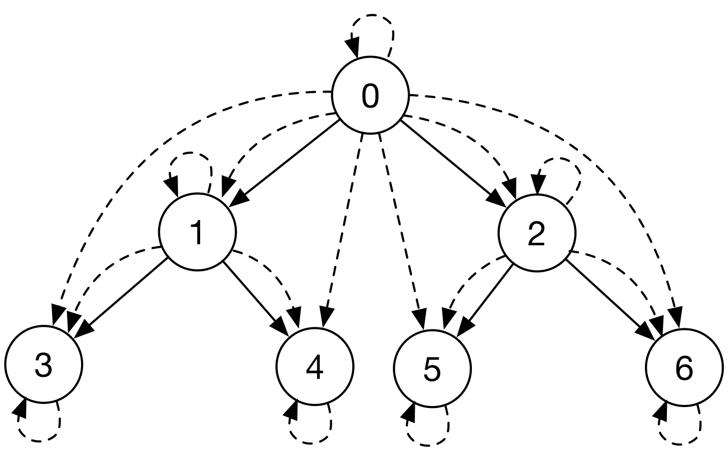
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$

$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$

$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$

$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$

... $\wedge$

$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6) \wedge$

$path(0,3) \vee \neg path(0,1) \vee \neg edge(1,3) \wedge$

... $\wedge$

$path(0,6) \vee \neg path(0,2) \vee \neg edge(2,6)$

Soft clauses:

$(\neg path(0,0)$ **weight 1.5**$) \wedge$

... $\wedge$

$(\neg path(6,6)$ **weight 1.5**$) \wedge$

$(\neg path(0,1)$ **weight 1.5**$) \wedge$

... $\wedge$

$(\neg path(2,6)$ **weight 1.5**$)$

Example: Iteration 4 - Check

Input relations:

edge(x, y)

Output relations:

path(x, y)

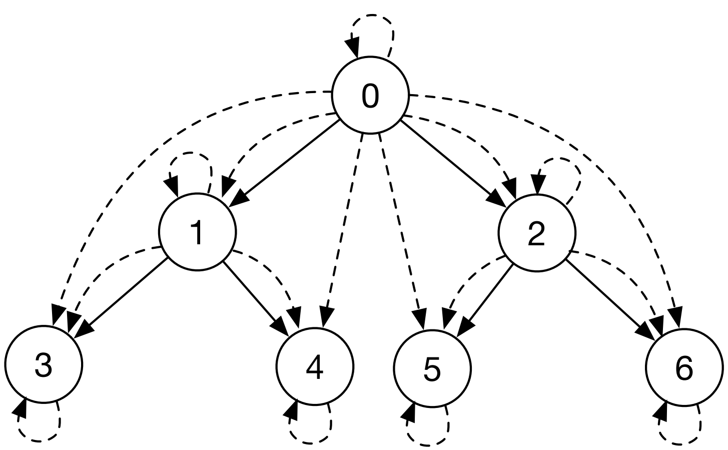
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$$
$$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$$
$$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$$
$$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$$
$$\ldots \wedge$$
$$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6) \wedge$$
$$path(0,3) \vee \neg path(0,1) \vee \neg edge(1,3) \wedge$$
$$\ldots \wedge$$
$$path(0,6) \vee \neg path(0,2) \vee \neg edge(2,6)$$

Soft clauses:

$(\neg path(0,0)$ **weight 1.5**$) \wedge$

$\ldots \wedge$

$(\neg path(6,6)$ **weight 1.5**$) \wedge$

$(\neg path(0,1)$ **weight 1.5**$) \wedge$

$\ldots \wedge$

$(\neg path(2,6)$ **weight 1.5**$)$

Example: Iteration 5 - Solve

Input relations:

edge(x, y)

Output relations:

path(x, y)

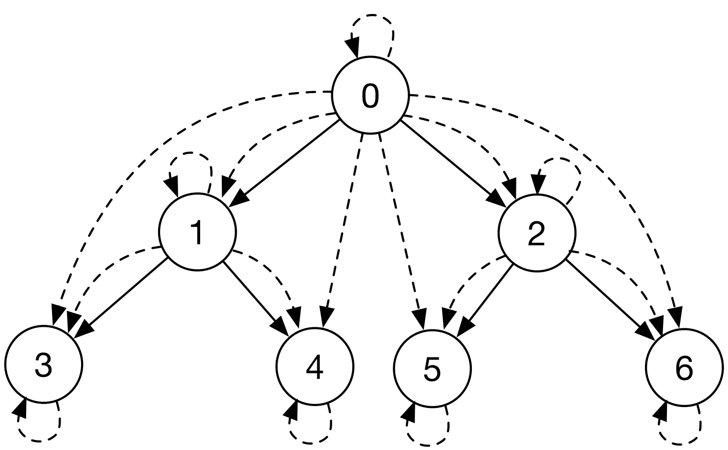
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$

$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$

$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$

$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$

... $\wedge$

$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6) \wedge$

$path(0,3) \vee \neg path(0,1) \vee \neg edge(1,3) \wedge$

... $\wedge$

$path(0,6) \vee \neg path(0,2) \vee \neg edge(2,6)$

Soft clauses:

($\neg path(0,0)$ **weight 1.5**) $\wedge$

... $\wedge$

($\neg path(6,6)$ **weight 1.5**) $\wedge$

($\neg path(0,1)$ **weight 1.5**) $\wedge$

... $\wedge$

($\neg path(2,6)$ **weight 1.5**) $\wedge$

($\neg path(0,3)$ **weight 1.5**) $\wedge$

... $\wedge$

($\neg path(0,6)$ **weight 1.5**)

Example: Iteration 5 - Check

1) All hard constraints are satisfied
2) No new violated soft constraints
=> sound to terminate

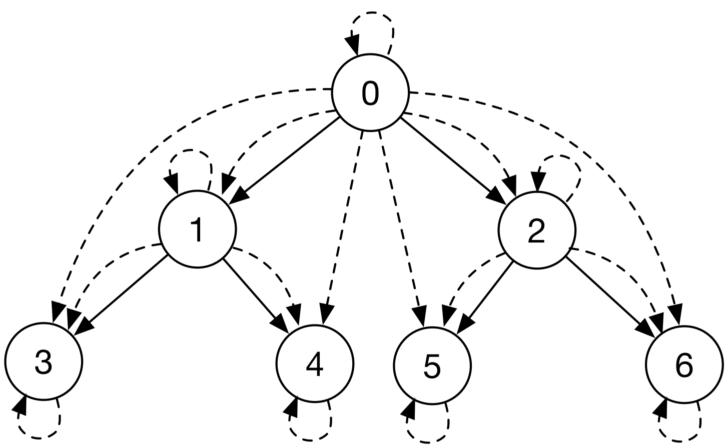
Hard constraints:

path(x, x).

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

$edge(0,1) \wedge \ldots \wedge edge(2,6) \wedge$
$path(0,0) \wedge \ldots \wedge path(6,6) \wedge$
$path(0,1) \vee \neg path(0,0) \vee \neg edge(0,1) \wedge$
$path(0,2) \vee \neg path(0,0) \vee \neg edge(0,2) \wedge$
... $\wedge$
$path(2,6) \vee \neg path(2,2) \vee \neg edge(2,6) \wedge$
$path(0,3) \vee \neg path(0,1) \vee \neg edge(1,3) \wedge$
... $\wedge$
$path(0,6) \vee \neg path(0,2) \vee \neg edge(2,6)$

Soft clauses:

($\neg path(0,0)$ **weight 1.5**) $\wedge$
... $\wedge$
($\neg path(6,6)$ **weight 1.5**) $\wedge$
($\neg path(0,1)$ **weight 1.5**) $\wedge$
... $\wedge$
($\neg path(2,6)$ **weight 1.5**) $\wedge$
($\neg path(0,3)$ **weight 1.5**) $\wedge$
... $\wedge$
($\neg path(0,6)$ **weight 1.5**)

Horn-Guided Optimization

Input relations:

edge(x, y)

Output relations:

path(x, y)

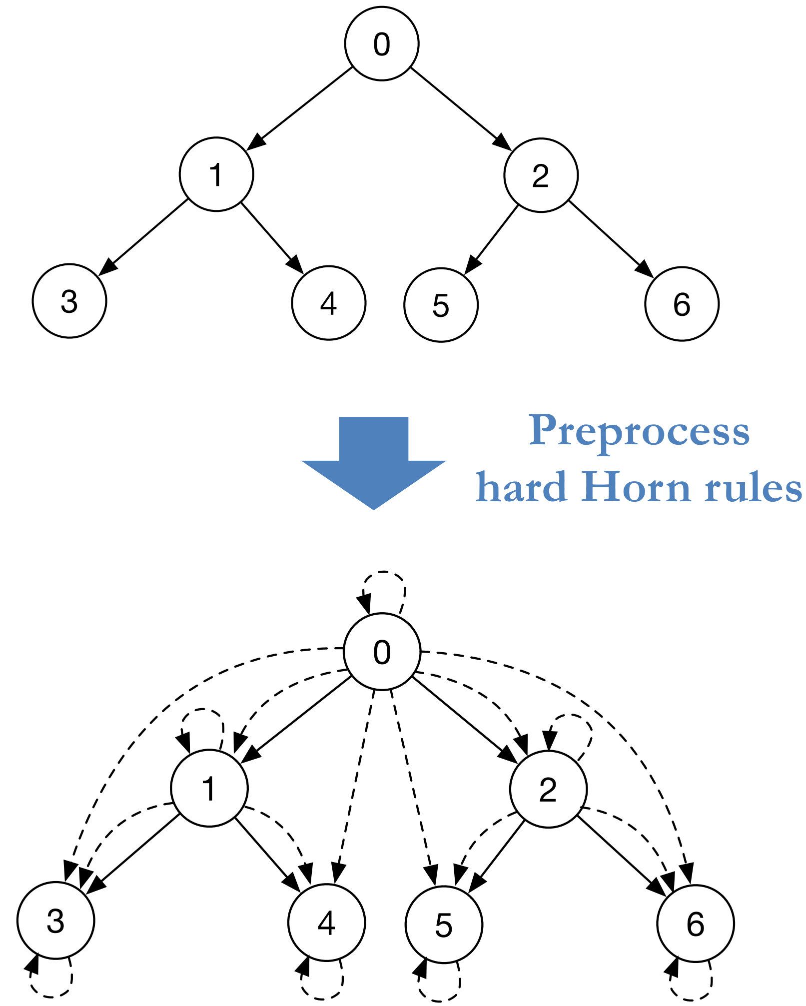
Hard constraints:

path(x, x). **Horn Rules!**

path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Horn-Guided Optimization

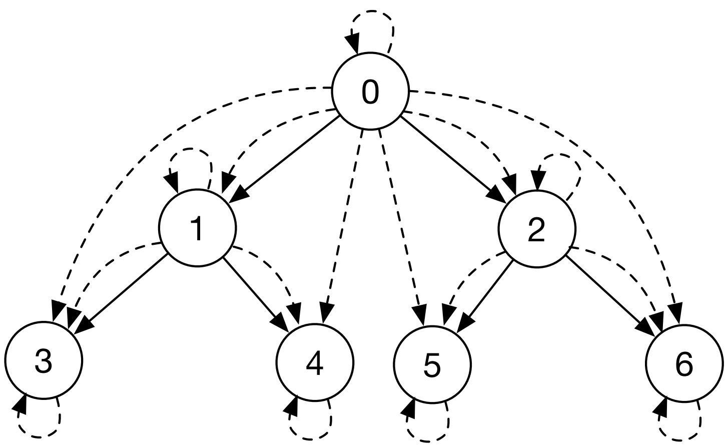
1) All hard constraints are satisfied
2) No new violated soft constraints
=> sound to terminate

Hard constraints:

path(x, x).
path(x, z) :- path(x, y), edge(y, z).

Soft constraints:

¬ path(x, y). **weight 1.5**



Hard clauses:

Soft clauses:

$(\neg path(0,0)$ **weight 1.5**$) \wedge$
... $\wedge$
$(\neg path(6,6)$ **weight 1.5**$) \wedge$
$(\neg path(0,1)$ **weight 1.5**$) \wedge$
... $\wedge$
$(\neg path(2,6)$ **weight 1.5**$) \wedge$
$(\neg path(0,3)$ **weight 1.5**$) \wedge$
... $\wedge$
$(\neg path(0,6)$ **weight 1.5**$)$

Performance Evaluation

	total # ground clauses	# iterations		total time (hours : mins)		# ground clauses	
		Lazy	Guided	Lazy	Guided	Lazy	Guided
avrora	1.8×10^{26}	492	12	6:31	0:25	0.8M	1.6M
ftp	3.7×10^{23}	463	5	7:53	0:08	1.2M	1.4M
hedc	1.9×10^{24}	354	6	1:55	0:06	0.8M	0.9M
luindex	1.6×10^{25}	481	7	4:07	0:12	0.6M	1.1M
lusearch	1.7×10^{25}	429	6	2:38	0:14	0.6M	1.0M
weblech	4.4×10^{24}	416	6	1:59	0:07	0.6M	0.9M

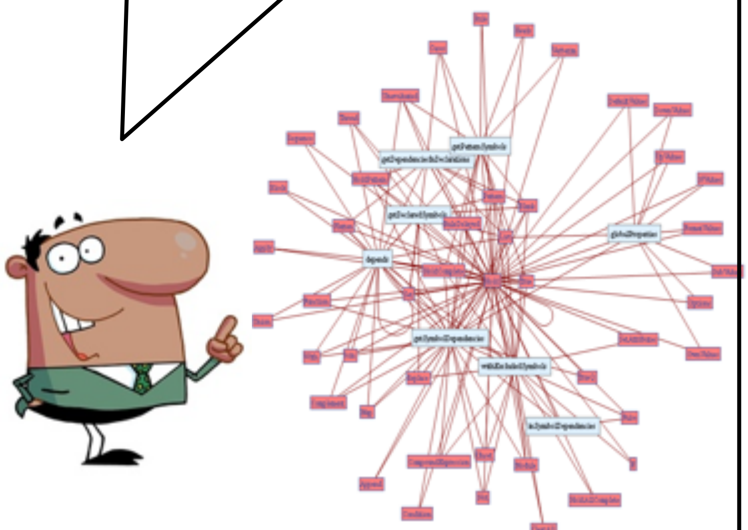
Overview of Techniques

- General Framework [SAT 2015]
- Bottom-Up Solving [AAAI 2016]
- Top-Down Solving [POPL 2016]

Queries in Different Domains

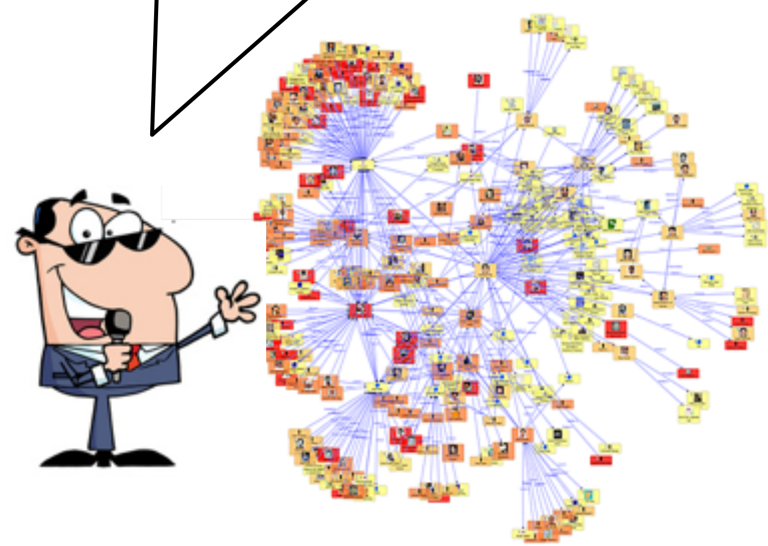
Program Reasoning:

Does variable **head** alias with variable **tail** on line 50 in Complex.java?



Information Retrieval:

Is **Dijkstra** most likely an author of "Structured Programming"?



Queries in MaxSAT

	$a \quad \wedge$	(C1)
	$\neg a \vee b \quad \wedge$	(C2)
4	$\neg b \vee c \quad \wedge$	(C3)
2	$\neg c \vee d \quad \wedge$	(C4)
7	$\neg d$	(C5)

QUERIES = {a, d}

Query-Guided Maximum Satisfiability (Q-MaxSAT)

	$a \wedge$	(C1)
	$\neg a \vee b \wedge$	(C2)
4	$\neg b \vee c \wedge$	(C3)
2	$\neg c \vee d \wedge$	(C4)
7	$\neg d$	(C5)

QUERIES = {a, d}

Q-MaxSAT:

Given a MaxSAT formula φ and a set of queries $Q \subseteq V$, a solution to the Q-MaxSAT instance (φ, Q) is a partial solution $\alpha_Q : Q \rightarrow \{0, 1\}$ such that

$$\exists \alpha \in MaxSAT(\varphi). (\forall v \in Q. \alpha_Q = \alpha(v))$$

Query-Guided Maximum Satisfiability (Q-MaxSAT)

	a	$\wedge$	(C1)
	$\neg a \vee b$	$\wedge$	(C2)
4	$\neg b \vee c$	$\wedge$	(C3)
2	$\neg c \vee d$	$\wedge$	(C4)
7	$\neg d$		(C5)

QUERIES = {a, d}

Q-MaxSAT:

Given a MaxSAT formula φ and a set of queries $Q \subseteq V$, a solution to the Q-MaxSAT instance (φ, Q) is a partial solution $\alpha_Q : Q \rightarrow \{0, 1\}$ such that

$$\exists \alpha \in MaxSAT(\varphi). (\forall v \in Q. \alpha_Q = \alpha(v))$$

Solution: a = true, d = false

Query-Guided Maximum Satisfiability (Q-MaxSAT)

	$a \wedge$	(C1)
	$\neg a \vee b \wedge$	(C2)
4	$\neg b \vee c \wedge$	(C3)
2	$\neg c \vee d \wedge$	(C4)
7	$\neg d$	(C5)

QUERIES = {a, d}

Q-MaxSAT:

Given a MaxSAT formula φ and a set of queries $Q \subseteq V$, a solution to the Q-MaxSAT instance (φ, Q) is a partial solution $\alpha_Q : Q \rightarrow \{0, 1\}$ such that

$$\exists \alpha \in MaxSAT(\varphi).(\forall v \in Q.\alpha_Q = \alpha(v))$$

MaxSAT Solution: a = true, b = true, c = true, **d = false**

Our key idea:

Use a small set of clauses to succinctly summarize effect of unexplored clauses

Example

Queries = {v6}, formula =

```
v4                  weight 100  ∧
v8                  weight 100  ∧
¬ v7                weight 100  ∧
¬ v3  ∨  v1         weight 5    ∧
¬ v5  ∨  v2         weight 5    ∧
¬ v5  ∨  v3         weight 5    ∧
¬ v6  ∨  v5         weight 5    ∧
¬ v6  ∨  v7         weight 5    ∧
¬ v4  ∨  v6         weight 5    ∧
¬ v8  ∨  v6         weight 5    ∧
...
```

Example

Queries = {v6}, formula =

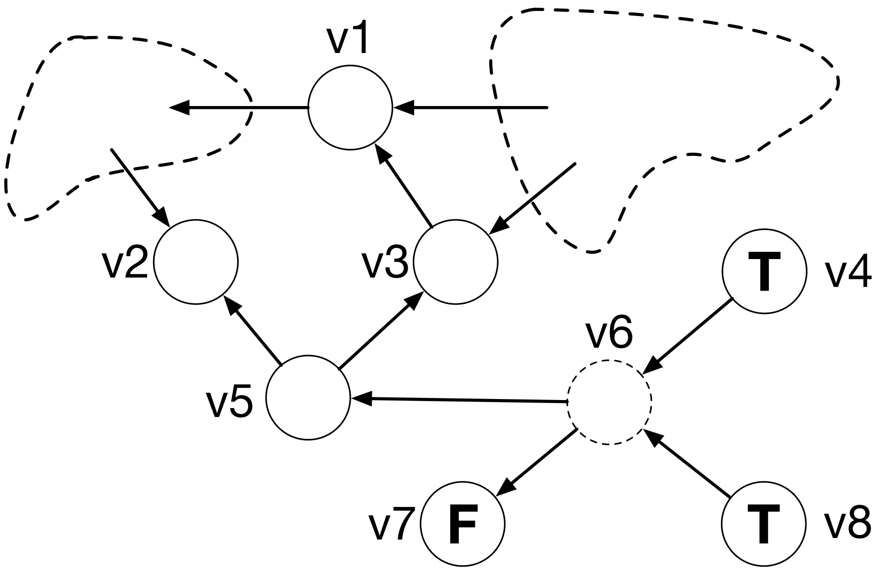
```
v4                    weight 100  ∧
v8                    weight 100  ∧
¬ v7                  weight 100  ∧
¬ v3  ∨  v1           weight 5    ∧
¬ v5  ∨  v2           weight 5    ∧
¬ v5  ∨  v3           weight 5    ∧
¬ v6  ∨  v5           weight 5    ∧
¬ v6  ∨  v7           weight 5    ∧
¬ v4  ∨  v6           weight 5    ∧
¬ v8  ∨  v6           weight 5    ∧
...
```

Example

Queries = {v6}, formula =

```
v4                  weight 100  ∧
v8                  weight 100  ∧
¬ v7                weight 100  ∧
¬ v3  ∨  v1         weight 5    ∧
¬ v5  ∨  v2         weight 5    ∧
¬ v5  ∨  v3         weight 5    ∧
¬ v6  ∨  v5         weight 5    ∧
¬ v6  ∨  v7         weight 5    ∧
¬ v4  ∨  v6         weight 5    ∧
¬ v8  ∨  v6         weight 5    ∧
...
```

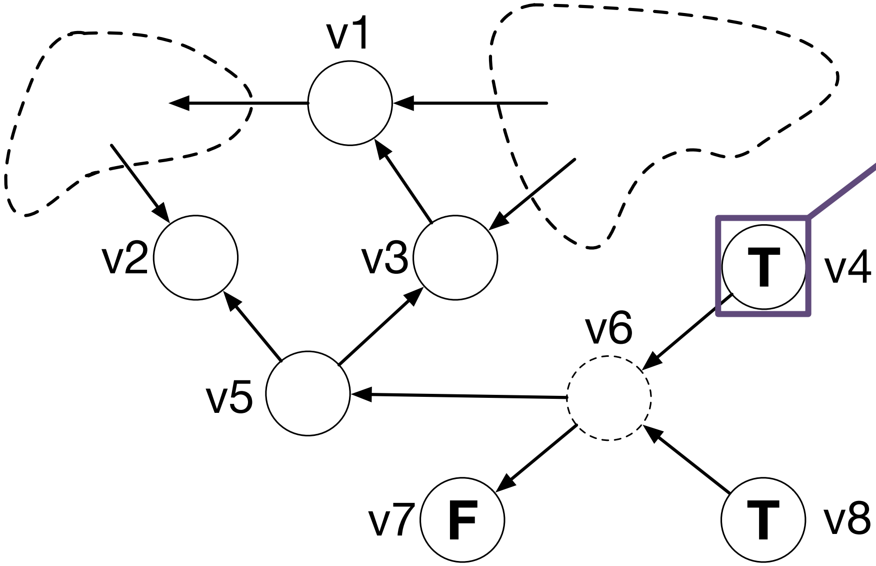
Example



Queries = {v6}, formula =

```
v4                 weight 100  ∧
v8                 weight 100  ∧
¬ v7               weight 100  ∧
¬ v3  ∨  v1        weight 5    ∧
¬ v5  ∨  v2        weight 5    ∧
¬ v5  ∨  v3        weight 5    ∧
¬ v6  ∨  v5        weight 5    ∧
¬ v6  ∨  v7        weight 5    ∧
¬ v4  ∨  v6        weight 5    ∧
¬ v8  ∨  v6        weight 5    ∧
...
```

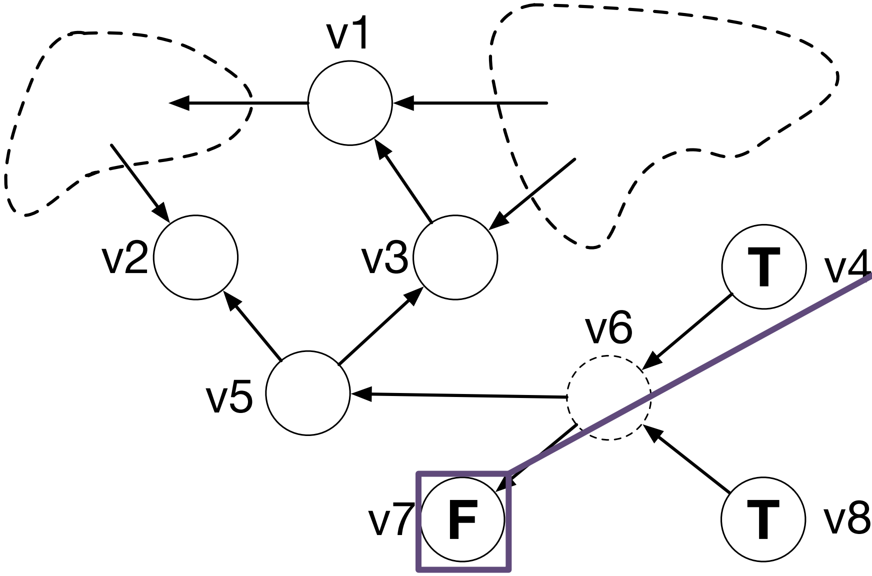
Example



Queries = {v6}, formula =

```
v4                weight 100  ∧
v8                weight 100  ∧
¬ v7              weight 100  ∧
¬ v3 ∨ v1         weight 5    ∧
¬ v5 ∨ v2         weight 5    ∧
¬ v5 ∨ v3         weight 5    ∧
¬ v6 ∨ v5         weight 5    ∧
¬ v6 ∨ v7         weight 5    ∧
¬ v4 ∨ v6         weight 5    ∧
¬ v8 ∨ v6         weight 5    ∧
...
```

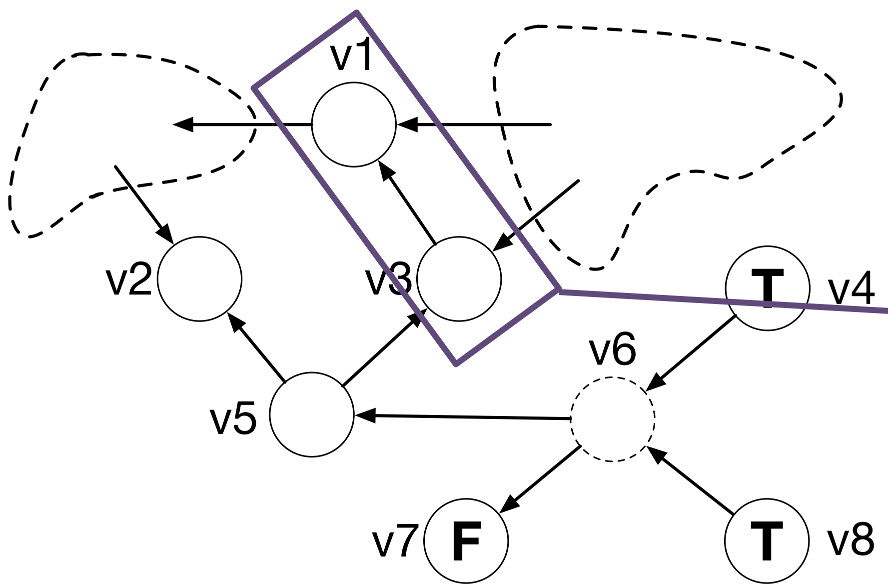
Example



Queries = {v6}, formula =

```
v4                weight 100  ∧
v8                weight 100  ∧
¬ v7              weight 100  ∧
¬ v3 ∨ v1         weight 5    ∧
¬ v5 ∨ v2         weight 5    ∧
¬ v5 ∨ v3         weight 5    ∧
¬ v6 ∨ v5         weight 5    ∧
¬ v6 ∨ v7         weight 5    ∧
¬ v4 ∨ v6         weight 5    ∧
¬ v8 ∨ v6         weight 5    ∧
...
```

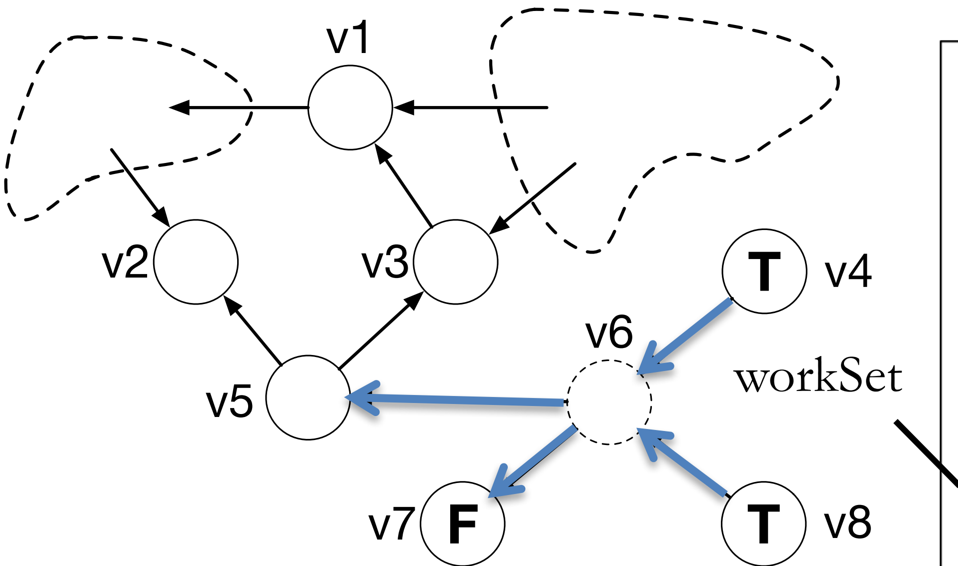
Example



Queries = {v6}, formula =

v4			weight 100	∧
v8			weight 100	∧
¬ v7			weight 100	∧
¬ v3	∨	v1	weight 5	∧
¬ v5	∨	v2	weight 5	∧
¬ v5	∨	v3	weight 5	∧
¬ v6	∨	v5	weight 5	∧
¬ v6	∨	v7	weight 5	∧
¬ v4	∨	v6	weight 5	∧
¬ v8	∨	v6	weight 5	∧
...				

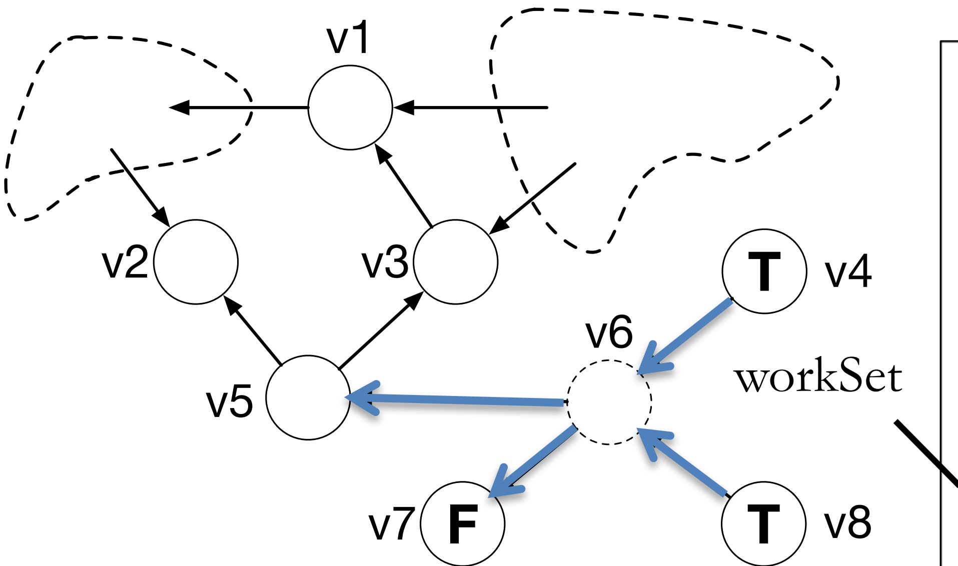
Example: Iteration 1



Queries = {v6}, formula =

v4		weight 100	∧
v8		weight 100	∧
¬ v7		weight 100	∧
¬ v3 ∨ v1		weight 5	∧
¬ v5 ∨ v2		weight 5	∧
¬ v5 ∨ v3		weight 5	∧
¬ v6 ∨ v5		weight 5	∧
¬ v6 ∨ v7		weight 5	∧
¬ v4 ∨ v6		weight 5	∧
¬ v8 ∨ v6		weight 5	∧
...			

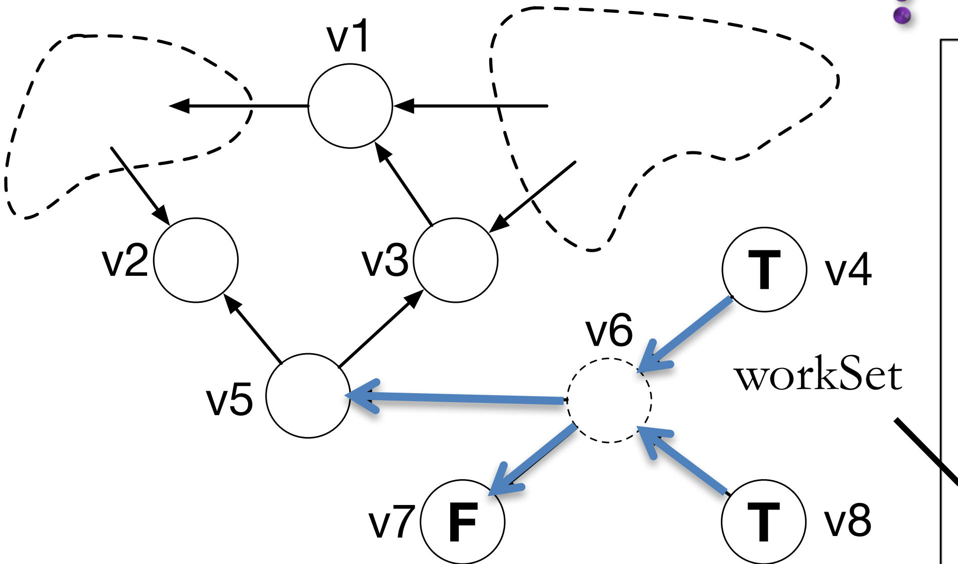
Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

v4			weight 100	∧
v8			weight 100	∧
¬ v7			weight 100	∧
¬ v3	∨	v1	weight 5	∧
¬ v5	∨	v2	weight 5	∧
¬ v5	∨	v3	weight 5	∧
¬ v6	∨	v5	weight 5	∧
¬ v6	∨	v7	weight 5	∧
¬ v4	∨	v6	weight 5	∧
¬ v8	∨	v6	weight 5	∧
...				

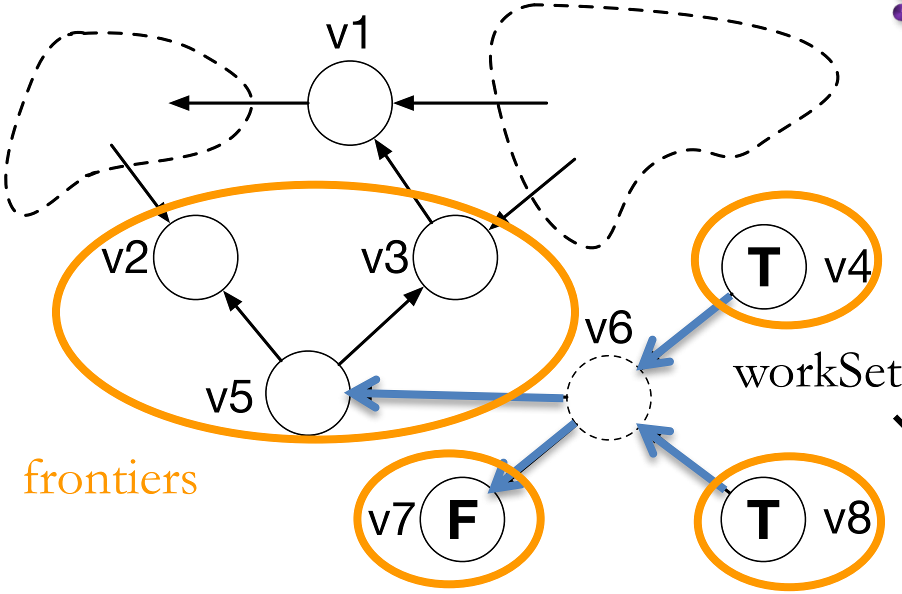
Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                  weight 100 ∧
v8                  weight 100 ∧
¬ v7                weight 100 ∧
¬ v3 ∨ v1           weight 5   ∧
¬ v5 ∨ v2           weight 5   ∧
¬ v5 ∨ v3           weight 5   ∧
¬ v6 ∨ v5           weight 5   ∧
¬ v6 ∨ v7           weight 5   ∧
¬ v4 ∨ v6           weight 5   ∧
¬ v8 ∨ v6           weight 5   ∧
...
```

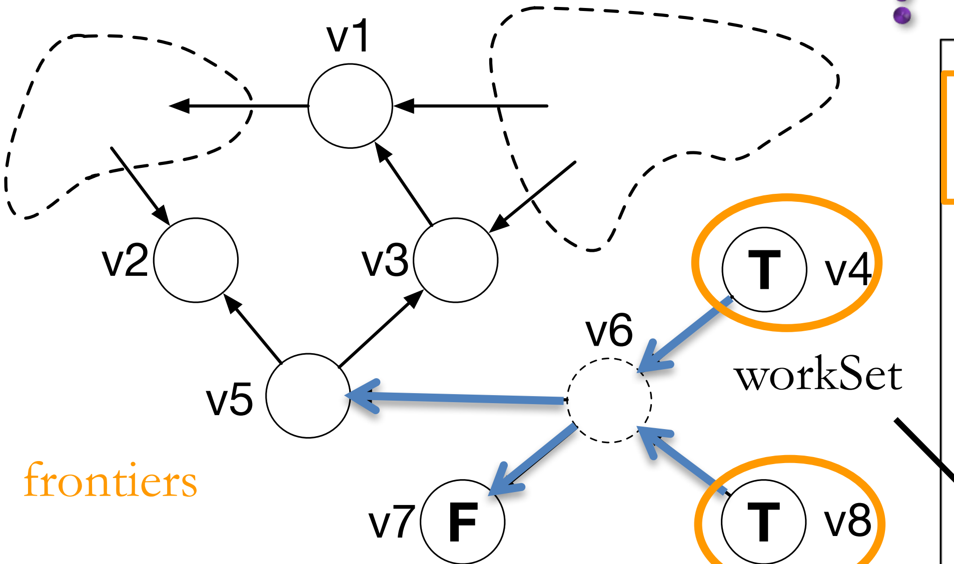
Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                 weight 100  ∧
v8                 weight 100  ∧
¬ v7               weight 100  ∧
¬ v3  ∨  v1        weight 5    ∧
¬ v5  ∨  v2        weight 5    ∧
¬ v5  ∨  v3        weight 5    ∧
¬ v6  ∨  v5        weight 5    ∧
¬ v6  ∨  v7        weight 5    ∧
¬ v4  ∨  v6        weight 5    ∧
¬ v8  ∨  v6        weight 5    ∧
...
```

Example: Iteration 1 (blue = true, red = false)

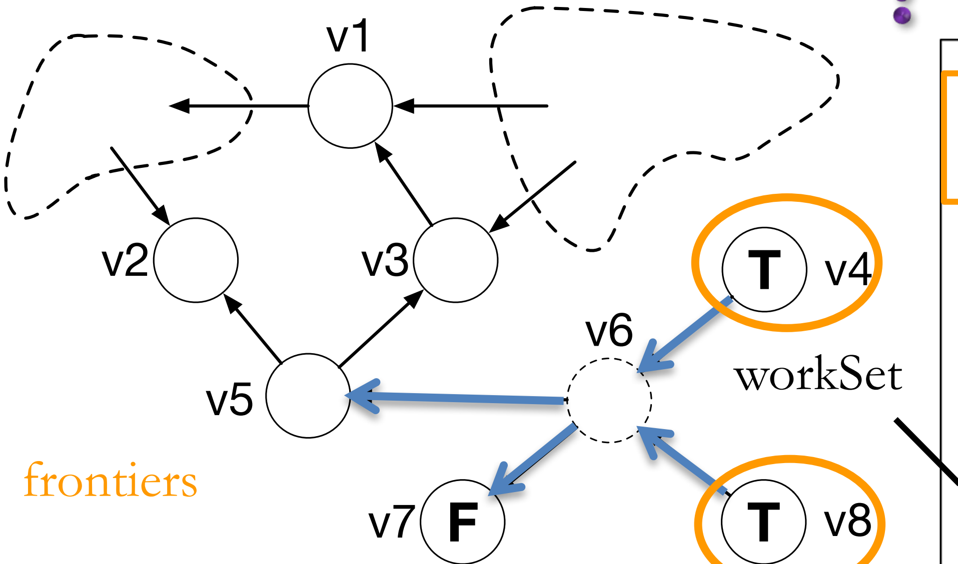


Queries = {v6}, formula =

v4			weight 100	$\wedge$
v8			weight 100	$\wedge$
$\neg$ v7			weight 100	$\wedge$
$\neg$ v3	$\vee$	v1	weight 5	$\wedge$
$\neg$ v5	$\vee$	v2	weight 5	$\wedge$
$\neg$ v5	$\vee$	v3	weight 5	$\wedge$
$\neg$ v6	$\vee$	v5	weight 5	$\wedge$
$\neg$ v6	$\vee$	v7	weight 5	$\wedge$
$\neg$ v4	$\vee$	v6	weight 5	$\wedge$
$\neg$ v8	$\vee$	v6	weight 5	$\wedge$
...				

Example: Iteration 1 (blue = true, red = false)

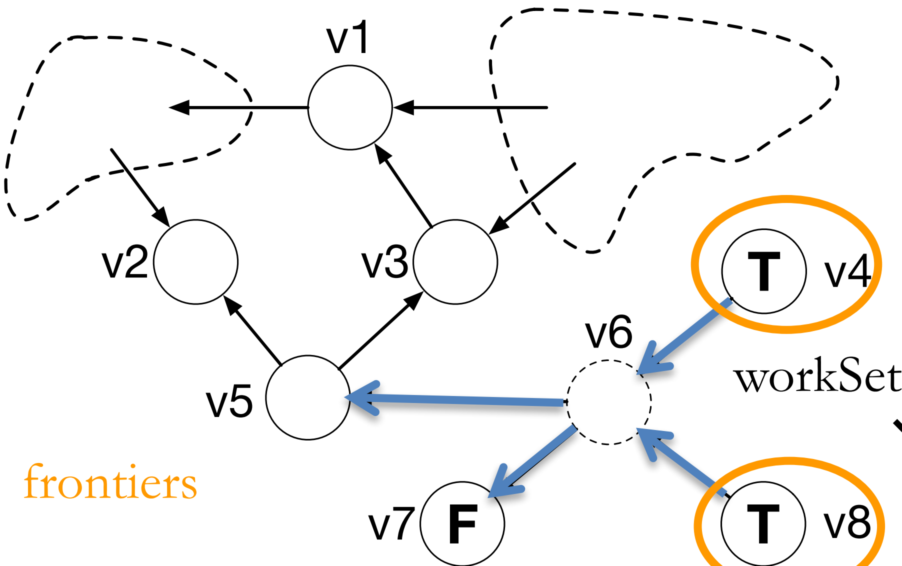
Queries = {v6}, formula =



	v4			weight 100	∧
	v8			weight 100	∧
¬	v7			weight 100	∧
¬	v3	∨	v1	weight 5	∧
¬	v5	∨	v2	weight 5	∧
¬	v5	∨	v3	weight 5	∧
¬	v6	∨	v5	weight 5	∧
¬	v6	∨	v7	weight 5	∧
¬	v4	∨	v6	weight 5	∧
¬	v8	∨	v6	weight 5	∧
...					

summarySet =
{(100, v4), (100, v8)}

Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

v4			weight 100 ∧
v8			weight 100 ∧
¬ v7			weight 100 ∧
¬ v3	∨	v1	weight 5 ∧
¬ v5	∨	v2	weight 5 ∧
¬ v5	∨	v3	weight 5 ∧
¬ v6	∨	v5	weight 5 ∧
¬ v6	∨	v7	weight 5 ∧
¬ v4	∨	v6	weight 5 ∧
¬ v8	∨	v6	weight 5 ∧
...			

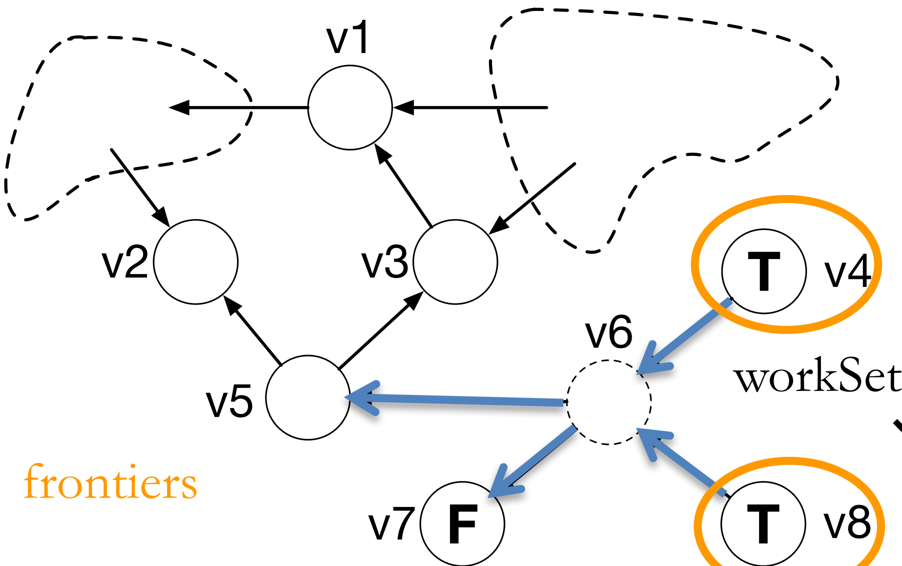
summarySet =
{(100, v4), (100, v8)}

20

220

max(workSet ∪ summarySet) - max(workSet) = 0 ?

Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

v4			**weight 100**	**∧**
v8			**weight 100**	**∧**
¬ **v7**			**weight 100**	**∧**
¬ **v3**	**∨**	**v1**	**weight 5**	**∧**
¬ **v5**	**∨**	**v2**	**weight 5**	**∧**
¬ **v5**	**∨**	**v3**	**weight 5**	**∧**
¬ **v6**	**∨**	**v5**	**weight 5**	**∧**
¬ **v6**	**∨**	**v7**	**weight 5**	**∧**
¬ **v4**	**∨**	**v6**	**weight 5**	**∧**
¬ **v8**	**∨**	**v6**	**weight 5**	**∧**
...				

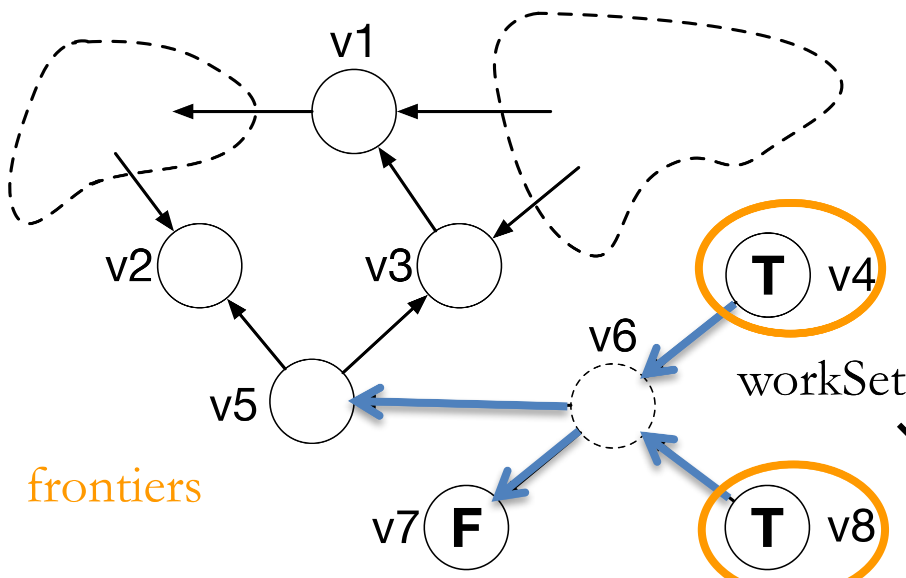
summarySet = {(100, v4), (100, v8)}

220

20

max(workSet ∪ summarySet) - max(workSet) = 0 ✖

Example: Iteration 1 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                 weight 100 ∧
v8                 weight 100 ∧
¬ v7               weight 100 ∧
¬ v3 ∨ v1          weight 5   ∧
¬ v5 ∨ v2          weight 5   ∧
¬ v5 ∨ v3          weight 5   ∧
¬ v6 ∨ v5          weight 5   ∧
¬ v6 ∨ v7          weight 5   ∧
¬ v4 ∨ v6          weight 5   ∧
¬ v8 ∨ v6          weight 5   ∧
```

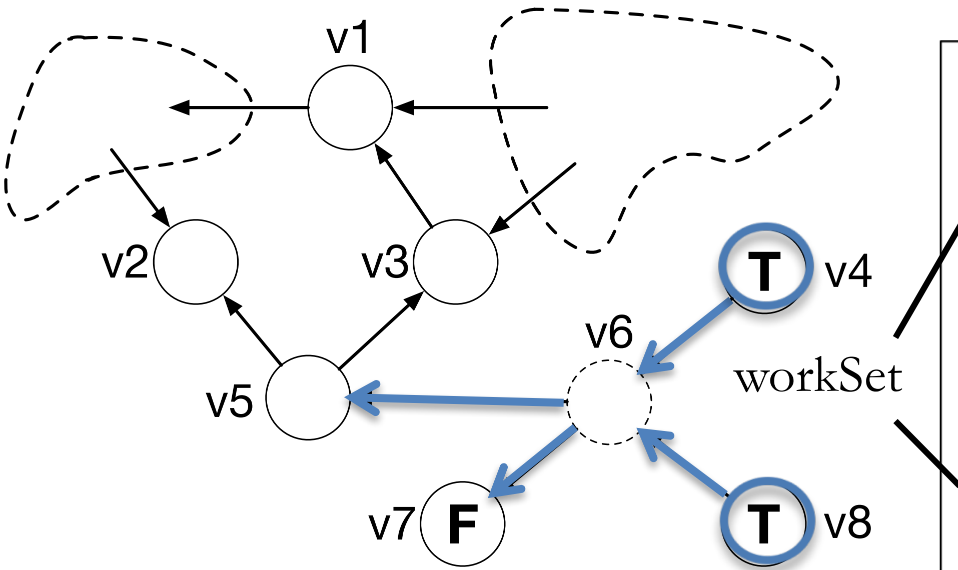
v4 = true, v5 = true, v6 = true, v7 = true, v8 = true

summarySet = {(100, v4), (100, v8)}

220

max(workSet ∪ summarySet) - max(workSet) = 0

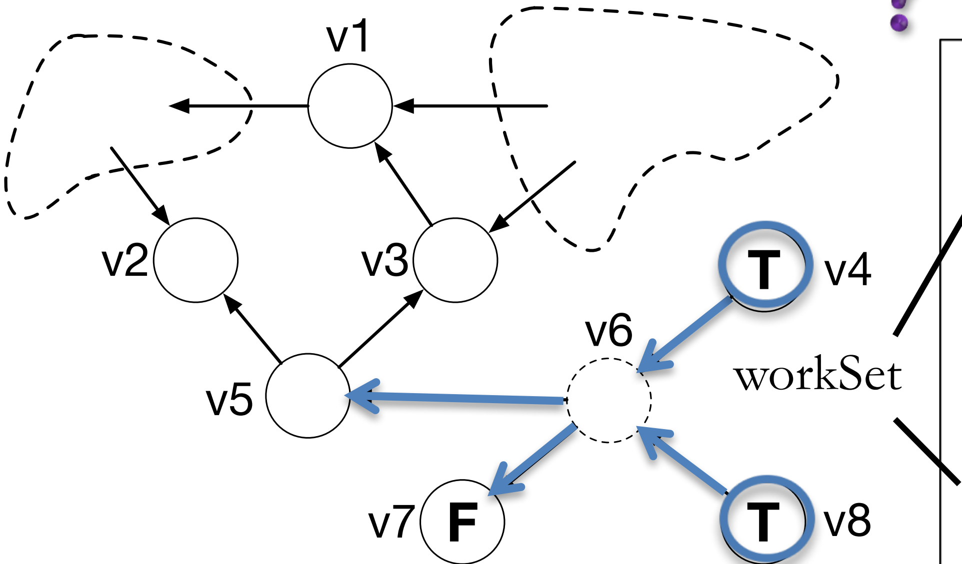
Example: Iteration 2 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                weight 100  ∧
v8                weight 100  ∧
¬ v7              weight 100  ∧
¬ v3  ∨  v1       weight 5    ∧
¬ v5  ∨  v2       weight 5    ∧
¬ v5  ∨  v3       weight 5    ∧
¬ v6  ∨  v5       weight 5    ∧
¬ v6  ∨  v7       weight 5    ∧
¬ v4  ∨  v6       weight 5    ∧
¬ v8  ∨  v6       weight 5    ∧
...
```

Example: Iteration 2 (blue = true, red = false)

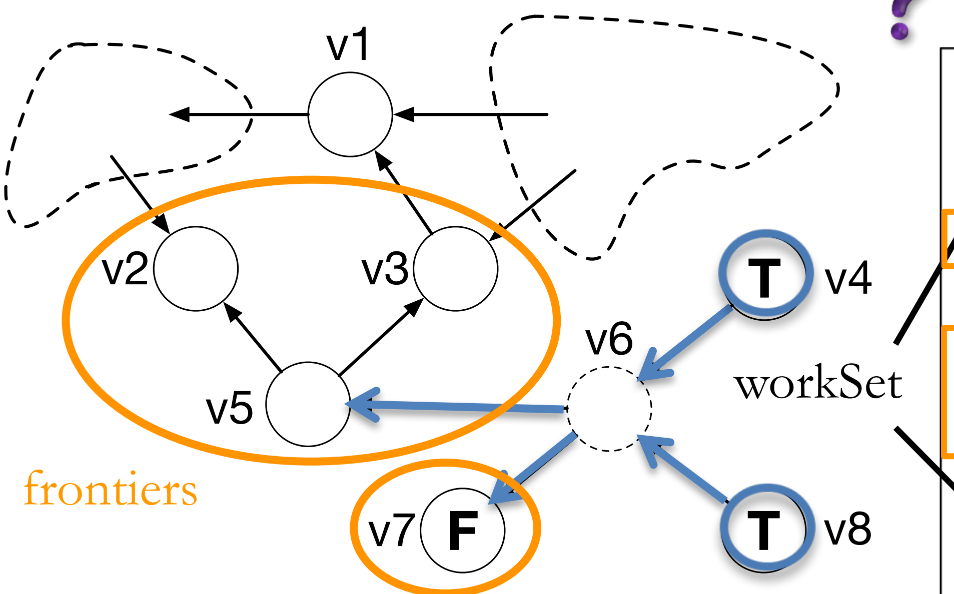


Queries = {v6}, formula =

	v4			weight 100	∧
	v8			weight 100	∧
¬	v7			weight 100	∧
¬	v3	∨	v1	weight 5	∧
¬	v5	∨	v2	weight 5	∧
¬	v5	∨	v3	weight 5	∧
¬	v6	∨	v5	weight 5	∧
¬	v6	∨	v7	weight 5	∧
¬	v4	∨	v6	weight 5	∧
¬	v8	∨	v6	weight 5	∧

...

Example: Iteration 2 (blue = true, red = false)

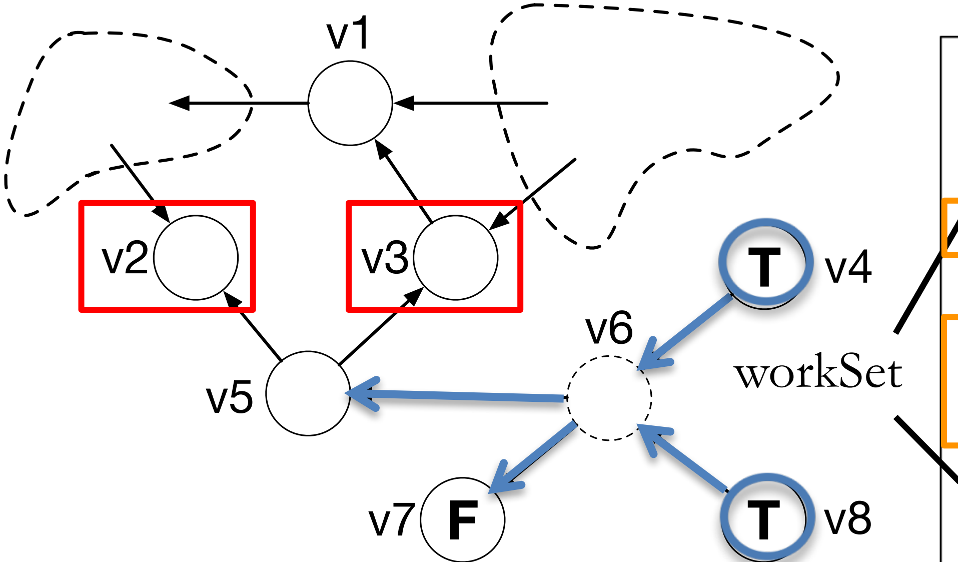


summarySet = {(100, ¬v7), (5, ¬v5 ∨ v2), (5, ¬v5 ∨ v3)}

Queries = {v6}, formula =

```
v4                weight 100  ∧
v8                weight 100  ∧
¬ v7              weight 100  ∧
¬ v3  ∨  v1       weight 5    ∧
¬ v5  ∨  v2       weight 5    ∧
¬ v5  ∨  v3       weight 5    ∧
¬ v6  ∨  v5       weight 5    ∧
¬ v6  ∨  v7       weight 5    ∧
¬ v4  ∨  v6       weight 5    ∧
¬ v8  ∨  v6       weight 5    ∧
...
```

Example: Iteration 2 (blue = true, red = false)



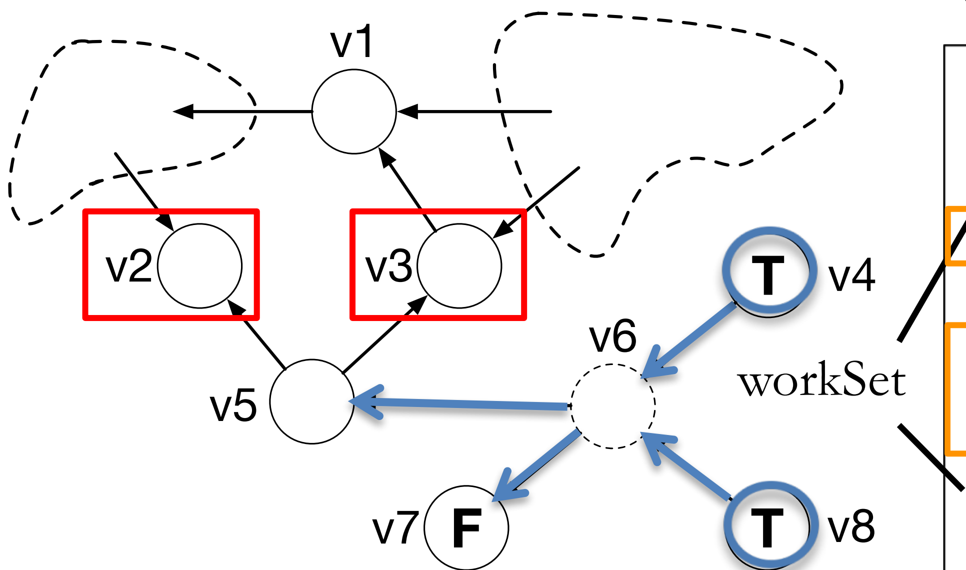
summarySet = {(100, ¬v7), (5, ¬v5 ∨ v2), (5, ¬v5 ∨ v3)}

Queries = {v6}, formula =

```
v4              weight 100  ∧
v8              weight 100  ∧
¬ v7            weight 100  ∧
¬ v3  ∨  v1     weight 5    ∧
¬ v5  ∨  v2     weight 5    ∧
¬ v5  ∨  v3     weight 5    ∧
¬ v6  ∨  v5     weight 5    ∧
¬ v6  ∨  v7     weight 5    ∧
¬ v4  ∨  v6     weight 5    ∧
¬ v8  ∨  v6     weight 5    ∧
...
```

max(workSet ∪ summarySet) - max(workSet) = 0 ?

Example: Iteration 2 (blue = true, red = false)



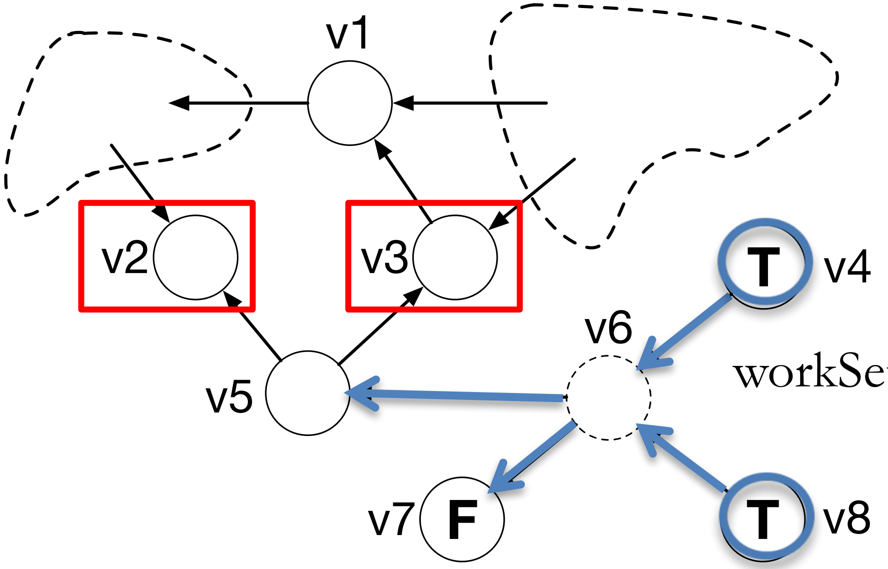
Queries = {v6}, formula =

```
v4                 weight 100  ∧
v8                 weight 100  ∧
¬ v7               weight 100  ∧
¬ v3  ∨  v1        weight 5    ∧
¬ v5  ∨  v2        weight 5    ∧
¬ v5  ∨  v3        weight 5    ∧
¬ v6  ∨  v5        weight 5    ∧
¬ v6  ∨  v7        weight 5    ∧
¬ v4  ∨  v6        weight 5    ∧
¬ v8  ∨  v6        weight 5    ∧
...
```

summarySet = {(100, ¬v7), (5, ¬v5), (5, ¬v5)}

max(workSet ∪ summarySet) - max(workSet) = 0 ?

Example: Iteration 2 (blue = true, red = false)



Queries = {v6}, formula =

workSet

```
v4                  weight 100  ∧
v8                  weight 100  ∧
¬ v7                weight 100  ∧
¬ v3  ∨  v1         weight 5    ∧
¬ v5  ∨  v2         weight 5    ∧
¬ v5  ∨  v3         weight 5    ∧
¬ v6  ∨  v5         weight 5    ∧
¬ v6  ∨  v7         weight 5    ∧
¬ v4  ∨  v6         weight 5    ∧
¬ v8  ∨  v6         weight 5    ∧
...
```

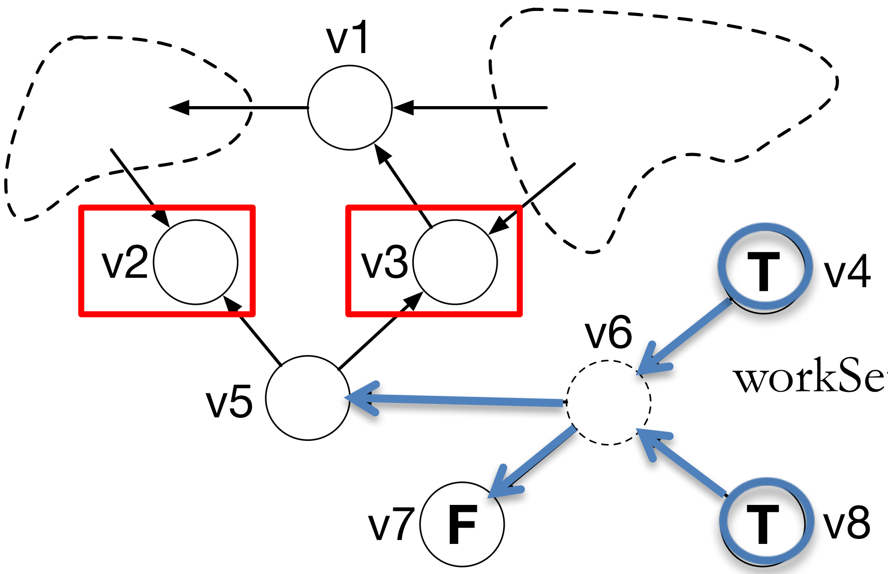
summarySet = {(100, ¬v7), (5, ¬v5), (5, ¬v5)}

220

320

max(workSet ∪ summarySet) - max(workSet) = 0 ✖

Example: Iteration 2 (blue = true, red = false)



workSet

Queries = {v6}, formula =

```
v4                      weight 100  ∧
v8                      weight 100  ∧
¬ v7                    weight 100  ∧
¬ v3  ∨  v1             weight 5    ∧
¬ v5  ∨  v2             weight 5    ∧
¬ v5  ∨  v3             weight 5    ∧
¬ v6  ∨  v5             weight 5    ∧
¬ v6  ∨  v7             weight 5    ∧
¬ v4  ∨  v6             weight 5    ∧
¬ v8  ∨  v6             weight 5    ∧
...
```

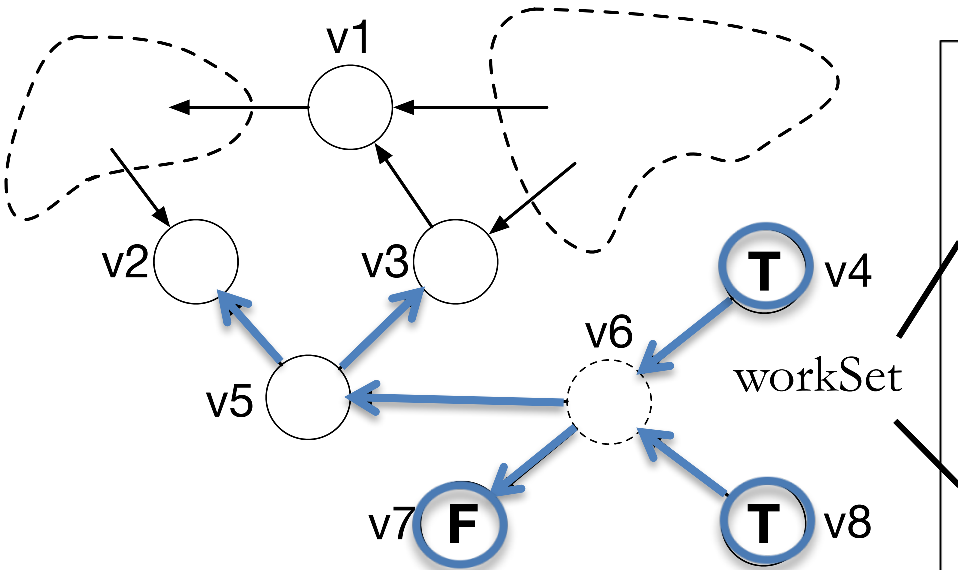
summarySet = {(100, ¬v7), (5, ¬v5), (5, ¬v5)}

320

max(workSet ∪ summarySet)

v4 = true, v5 = false, v6 = true, v7 = true, v8 = true

Example: Iteration 3 (blue = true, red = false)

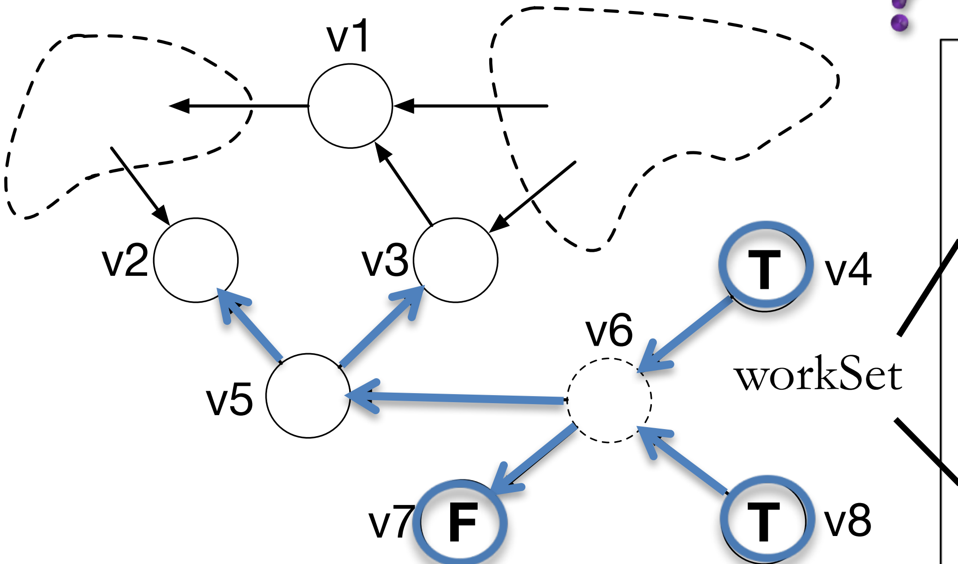


Queries = {v6}, formula =

	v4			weight 100	∧
	v8			weight 100	∧
¬	v7			weight 100	∧
¬	v3	∨	v1	weight 5	∧
¬	v5	∨	v2	weight 5	∧
¬	v5	∨	v3	weight 5	∧
¬	v6	∨	v5	weight 5	∧
¬	v6	∨	v7	weight 5	∧
¬	v4	∨	v6	weight 5	∧
¬	v8	∨	v6	weight 5	∧

...

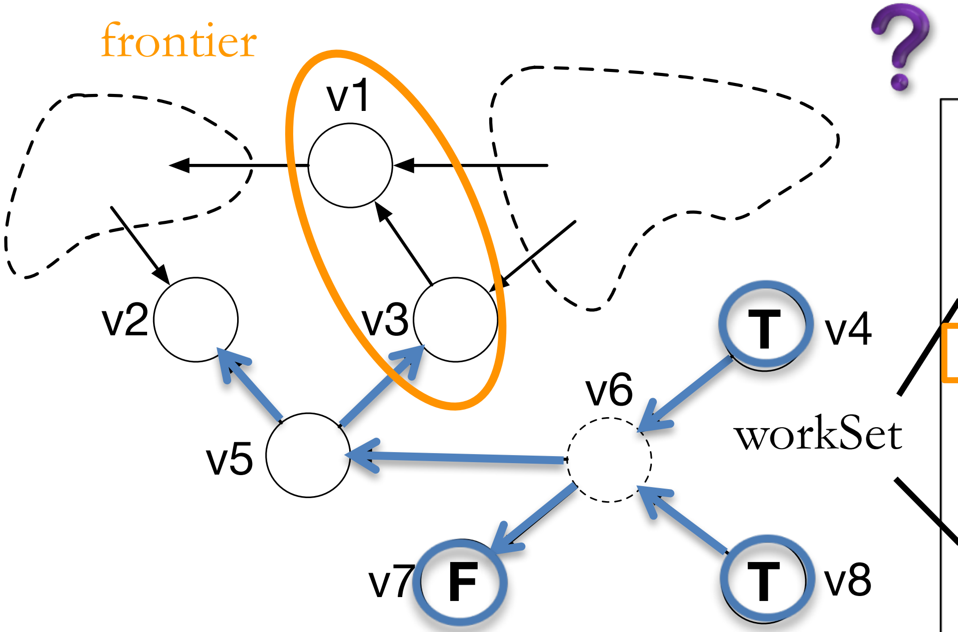
Example: Iteration 3 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                       weight 100  ∧
v8                       weight 100  ∧
¬ v7                     weight 100  ∧
¬ v3  ∨  v1              weight 5    ∧
¬ v5  ∨  v2              weight 5    ∧
¬ v5  ∨  v3              weight 5    ∧
¬ v6  ∨  v5              weight 5    ∧
¬ v6  ∨  v7              weight 5    ∧
¬ v4  ∨  v6              weight 5    ∧
¬ v8  ∨  v6              weight 5    ∧
...
```

Example: Iteration 3 (blue = true, red = false)

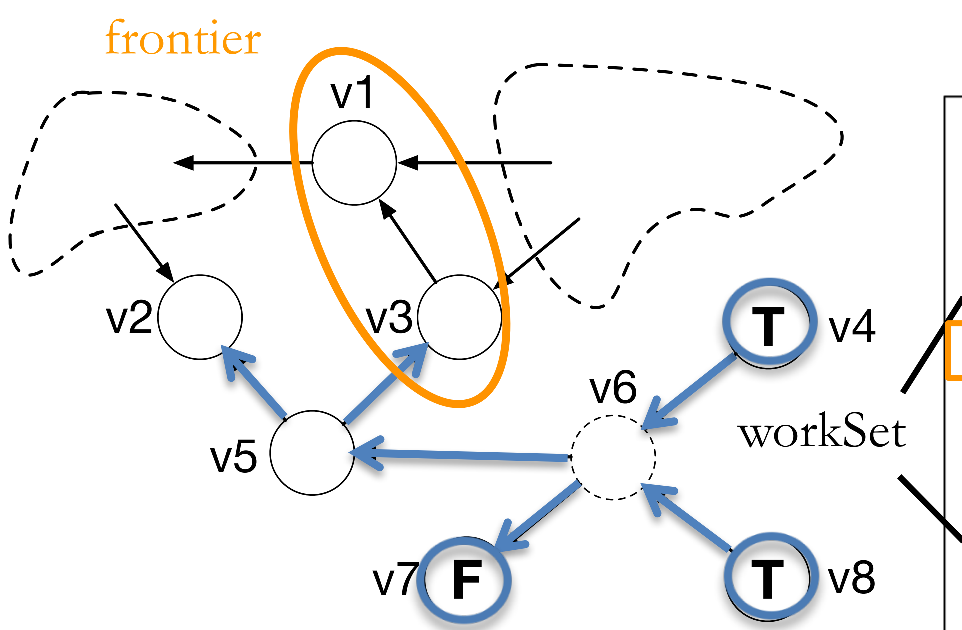


Queries = {v6}, formula =

```
v4                  weight 100  ∧
v8                  weight 100  ∧
¬ v7                weight 100  ∧
¬ v3  ∨  v1         weight 5    ∧
¬ v5  ∨  v2         weight 5    ∧
¬ v5  ∨  v3         weight 5    ∧
¬ v6  ∨  v5         weight 5    ∧
¬ v6  ∨  v7         weight 5    ∧
¬ v4  ∨  v6         weight 5    ∧
¬ v8  ∨  v6         weight 5    ∧
...
```

summarySet = $\{(5, \neg v3 \lor v1)\}$

Example: Iteration 3 (blue = true, red = false)



Queries = {v6}, formula =

	v4			weight 100	∧
	v8			weight 100	∧
¬	v7			weight 100	∧
¬	v3	∨	v1	weight 5	∧
¬	v5	∨	v2	weight 5	∧
¬	v5	∨	v3	weight 5	∧
¬	v6	∨	v5	weight 5	∧
¬	v6	∨	v7	weight 5	∧
¬	v4	∨	v6	weight 5	∧
¬	v8	∨	v6	weight 5	∧

...

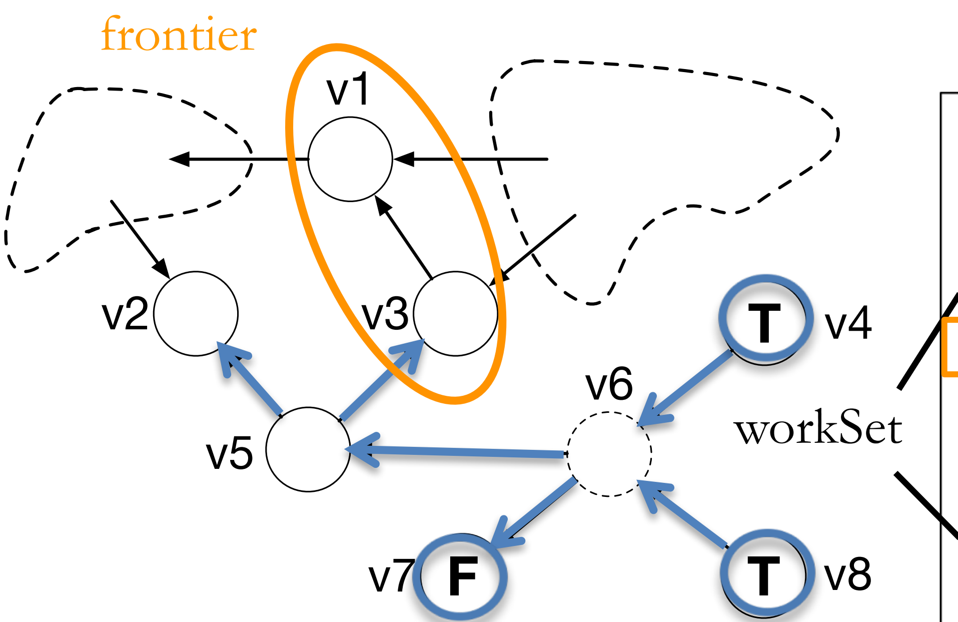
summarySet = {(5, ¬v3 ∨ v1)}

325

330

max(workSet ∪ summarySet) - max(workSet) = 0 ✖

Example: Iteration 3 (blue = true, red = false)



Queries = {v6}, formula =

```
v4                   weight 100  ∧
v8                   weight 100  ∧
¬ v7                 weight 100  ∧
¬ v3  ∨  v1          weight 5    ∧
¬ v5  ∨  v2          weight 5    ∧
¬ v5  ∨  v3          weight 5    ∧
¬ v6  ∨  v5          weight 5    ∧
¬ v6  ∨  v7          weight 5    ∧
¬ v4  ∨  v6          weight 5    ∧
¬ v8  ∨  v6          weight 5    ∧
...
```

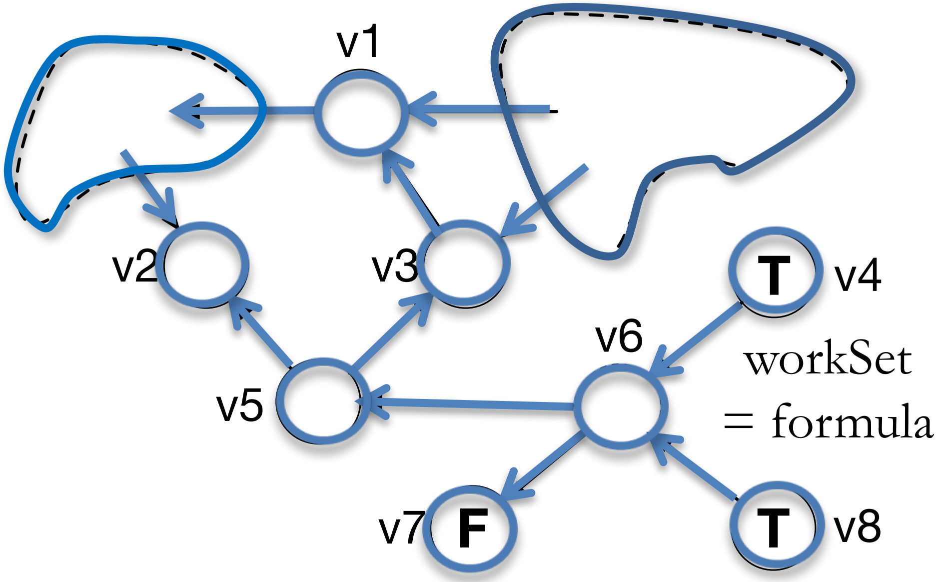
summarySet = {(5, ¬v3 ~~∨ v1)~~}

325

325

max(workSet ∪ summarySet) - max(workSet) = 0 ✓

Example



Queries = {v6}, formula =

```
v4                  weight 100  ∧
v8                  weight 100  ∧
¬ v7                weight 100  ∧
¬ v3  ∨  v1         weight 5    ∧
¬ v5  ∨  v2         weight 5    ∧
¬ v5  ∨  v3         weight 5    ∧
¬ v6  ∨  v5         weight 5    ∧
¬ v6  ∨  v7         weight 5    ∧
¬ v4  ∨  v6         weight 5    ∧
¬ v8  ∨  v6         weight 5    ∧
...
```

Benchmark Characteristics

	# queries	# variables	# clauses
ftp	55	2.3M	3M
hedc	36	3.8M	4.8M
weblech	25	5.8M	8.4M
lusearch	248	7.8M	10.9M
luindex	109	8.5M	11.9M
avrora	151	11.7M	16.3M
IE	6	47K	0.9M
ER	25	3K	4.8M
AR	10	0.3M	7.9M

K = thousands, M = millions

Performance Results

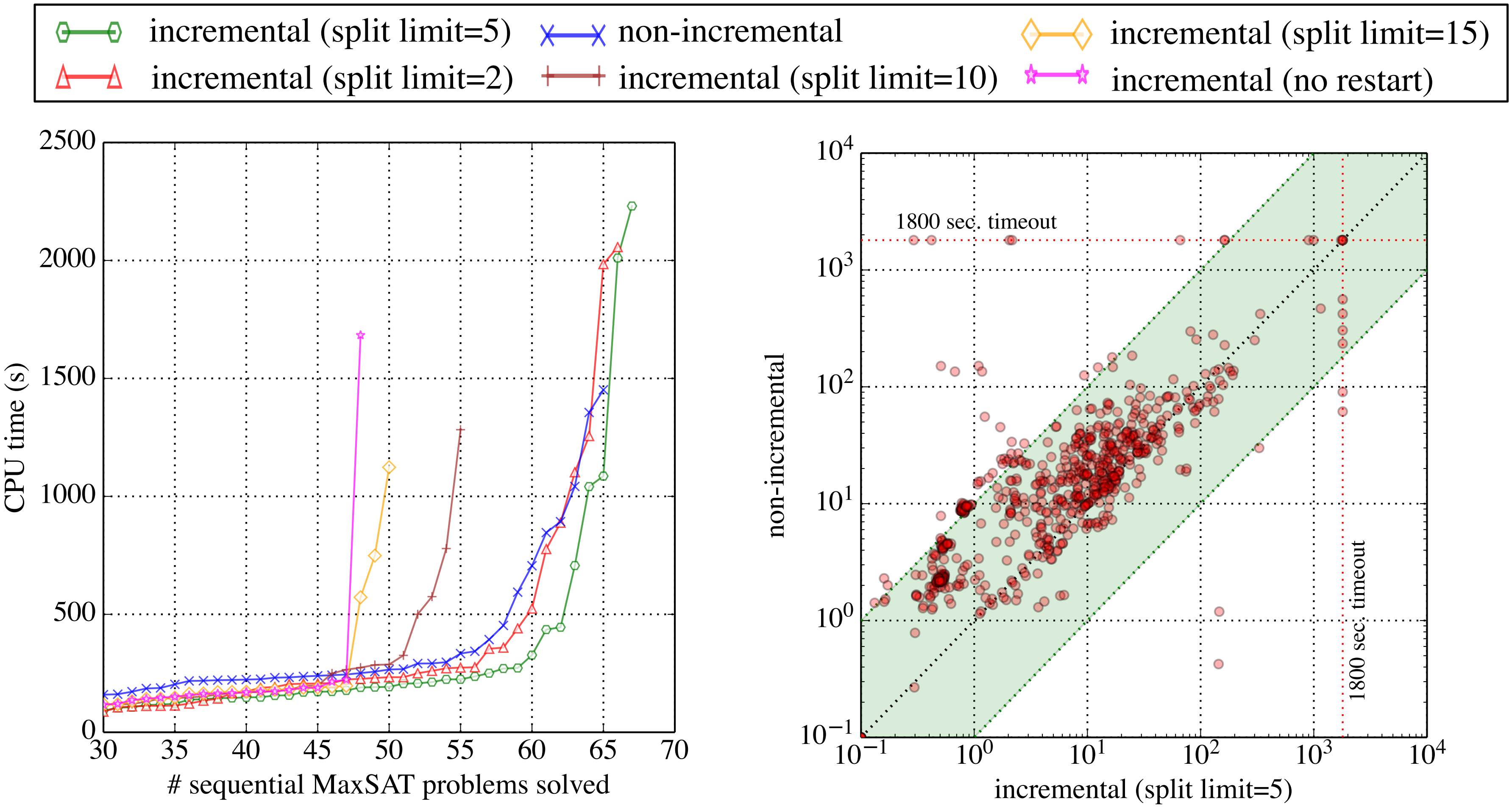
	running time (seconds)		peak memory (MB)		# clauses (M=million)	
	current	baseline	current	baseline	current	baseline
ftp	**16**	11	**16**	1,262	**0.03M**	3.0M
hedc	**23**	21	**181**	1,918	**0.4M**	4.8M
weblech	**4**	**timeout**	**363**	**timeout**	**0.9M**	8.4M
lusearch	**115**	**timeout**	**659**	**timeout**	**1.5M**	10.9M
luindex	**169**	**timeout**	**944**	**timeout**	**2.2M**	11.9M
avrora	**178**	**timeout**	**1,095**	**timeout**	**2.6M**	16.3M
IE	**2**	2,760	**13**	335	**27K**	0.9M
ER	**13**	2	**6**	44	**9K**	4.8M
AE	**4**	**timeout**	**4**	**timeout**	**2K**	7.9M

Incremental Solving [CP 2016]

φ_1 → $\varphi_2 = \varphi_1 \cup \Delta_1$ → $\varphi_3 = \varphi_2 \cup \Delta_2$ → $\cdots$

- Two levels of incrementality
  - MaxSAT level
    - Application solves a sequence of MaxSAT instances
    - Re-use unsat cores (for core-guided solver)
  - SAT level
    - Each MaxSAT solves a sequence of SAT instances
    - Leverage standard incremental SAT solving
- Key insight: sporadic restarts
  - Heuristically detect and avoid reusing bad unsat cores based on split-limit (max. # times a soft clause is split)

Performance Results

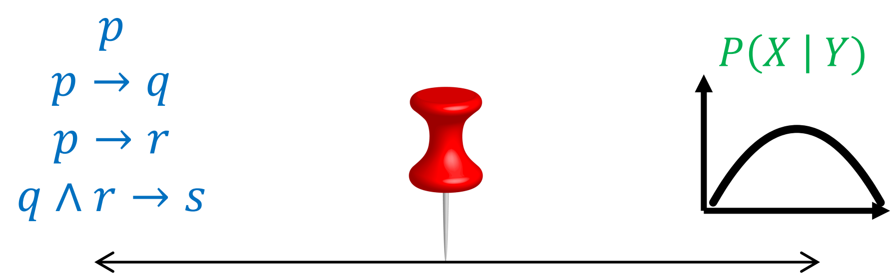


74 sequential MaxSAT problems (**669 individual** MaxSAT instances)

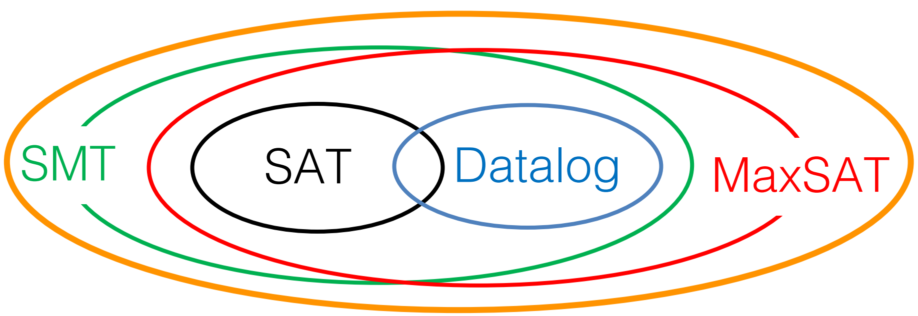
Future Directions

Humans In the Loop

Combining Logic With Probability



Optimization Solvers



MaxSMT
Markov Logic Network
Probabilistic Soft Logic
...

Data-Driven

GitHub

Bitbucket

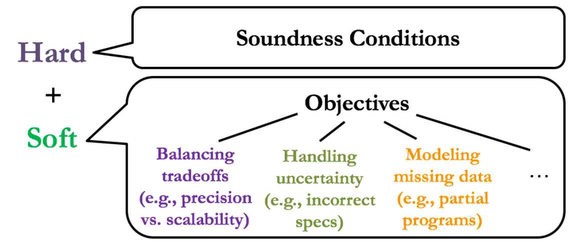
sourceforge

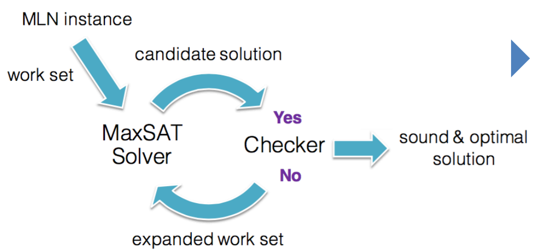
CVE

moz://a Bug Bounty Program

Conclusions

- New methodology to incorporate objectives into constraint-based software analyses





- General framework to solve weighted constraints that is sound, optimal and scalable

- Showed practical effectiveness for three dominant applications of software analyses