Points-to Analysis using BDDs

Marc Berndl, Ondřej Lhoták, Feng Qian, Laurie Hendren and Navindra Umanee
McGill University

Presented by Bruno Dufour
dufour@cs.rutgers.edu
Rutgers University DCS

Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions

Background – Points-to Analysis

- *Goal:* Given a (reference) variable v, find the set of objects to which v may point at runtime.
  - For each v, keep a set of possible objects (points-to set).

Background – Points-to Analysis

- *Goal:* Given a (reference) variable v, find the set of objects to which v may point at runtime.
  - For each v, keep a set of possible objects (points-to set).
- Problems

Background – Points-to Analysis

- *Goal:* Given a (reference) variable v, find the set of objects to which v may point at runtime.
  - For each v, keep a set of possible objects (points-to set).
- Problems
  - Large points-to sets

Background – Points-to Analysis

- *Goal:* Given a (reference) variable v, find the set of objects to which v may point at runtime.
  - For each v, keep a set of possible objects (points-to set).
- Problems
  - Large points-to sets
  - Large number of points-to sets

Background – Points-to Analysis

- *Goal:* Given a (reference) variable v, find the set of objects to which v may point at runtime.
  - For each v, keep a set of possible objects (points-to set).
- Problems
  - Large points-to sets
    $\longrightarrow$ Find efficient set representations
  - Large number of points-to sets
    $\longrightarrow$ Collapse equivalent variables

Points-to Example Code

```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
         a = b;
         b = a;
         c = b;
```

Points-to set: { }

Points-to Example Code

```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
      a = b;
      b = a;
      c = b;
```

Points-to set: { (a,X) (b,Y) (c,Z) }

Points-to Example Code

```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
        a = b;
        b = a;
        c = b;
```

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) }

Points-to Example Code

```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
      a = b;
      b = a;
      c = b;
```

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) }

Points-to Example Code

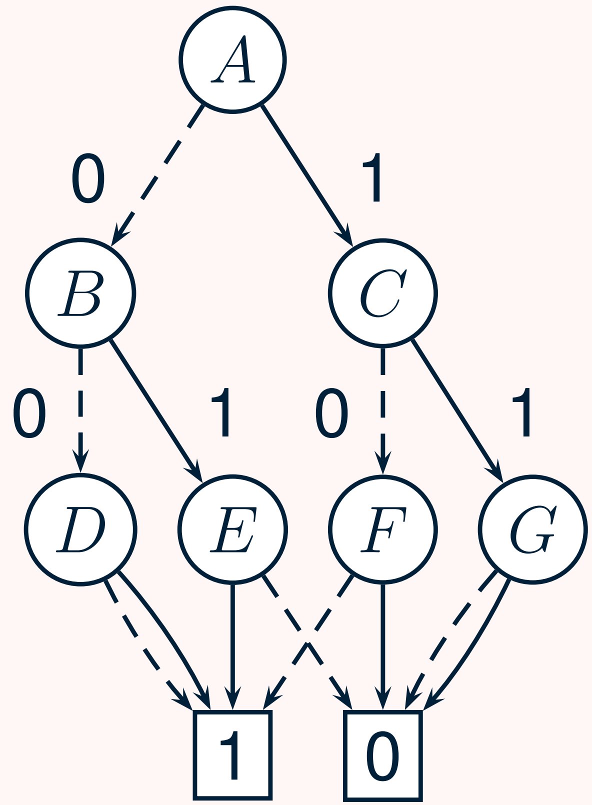
```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
        a = b;
        b = a;
        c = b;
```

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) (c,X) (c,Y) }

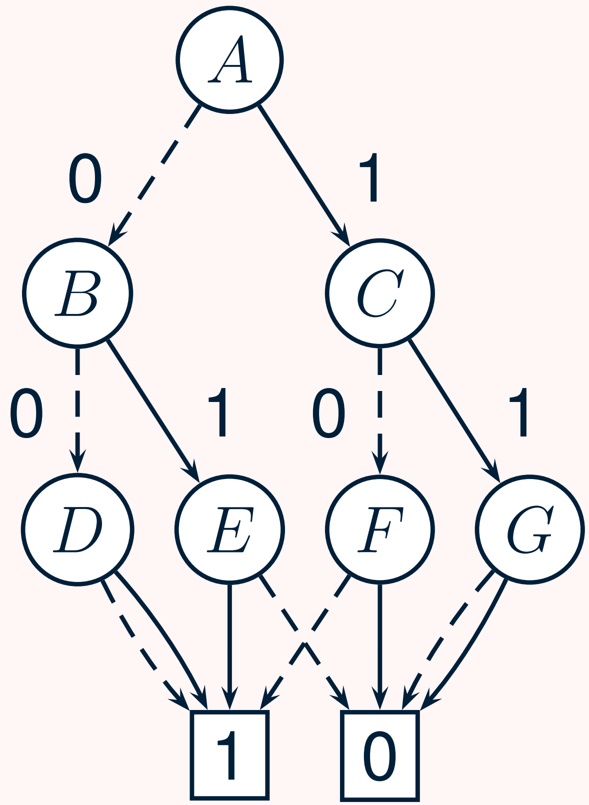
Background – BDDs

- Binary Decision Diagrams (BDDs) are data structures that are used to represent large sets with similarities.
- Introduced in [Bryant86]
- Applications in model checking
- Essentially single-root DAGs with out-degree two for each non-leaf node
- Some possible interpretations:
  - Set of binary strings
  - Representation of a boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$
  - Finite automaton with accepting state $\boxed{1}$ and rejecting state $\boxed{0}$ taking binary strings as input

Example BDD

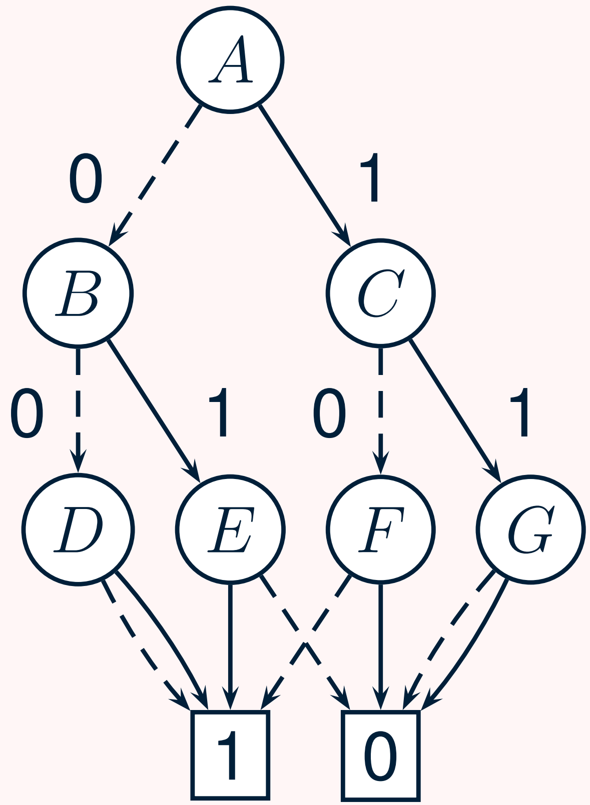


Example BDD

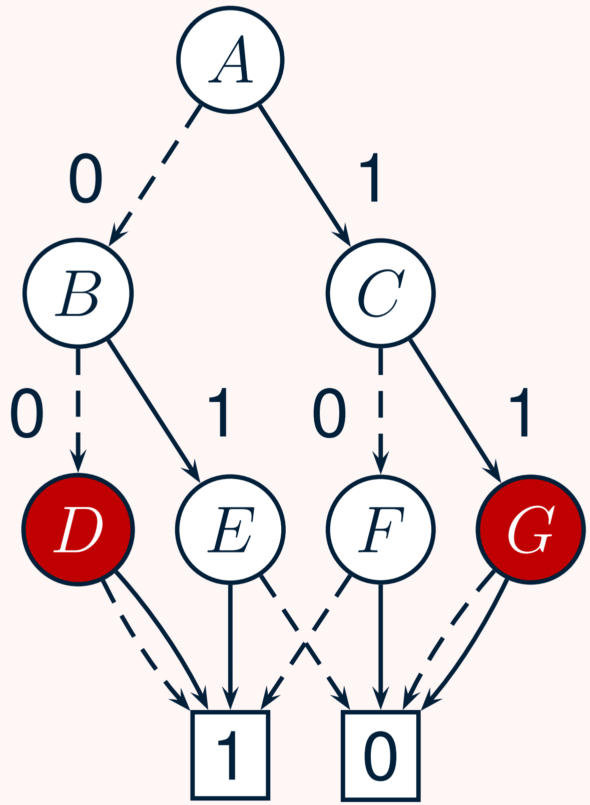


L = {000,001,011,100}

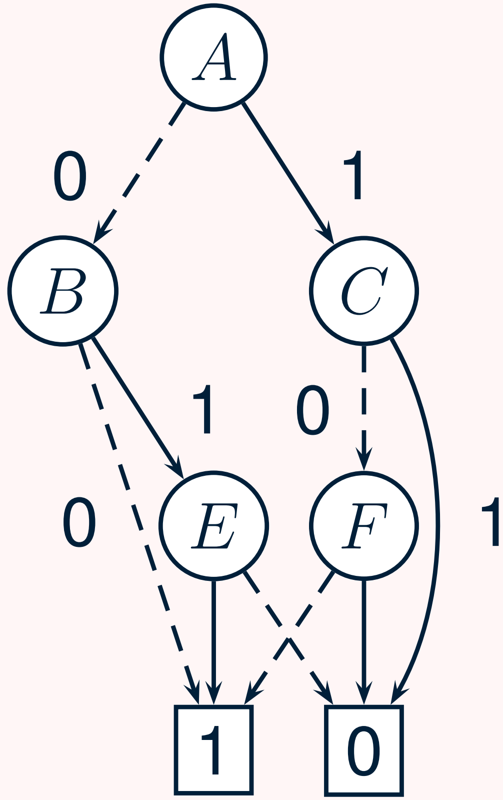
Reducing a BDD



Reducing a BDD



Reduced BDD

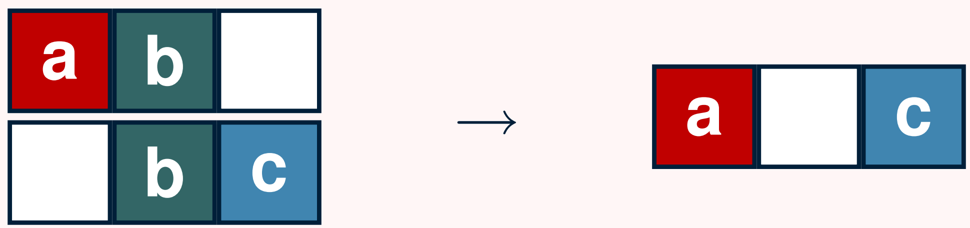


Types of BDDs

- Ordered BDDs (OBDDs)
  - variables are *ordered*
  - Each variable appears only in one level of the BDD
- Reduced Ordered BDDs (ROBDDs)
  - OBDDs in reduced form
  - Consistent ordering of nodes ensures uniqueness

BDD Operations

- BDDS support common set operations ($\cap$, $\cup$, ...)
- Existential quantification: $S = \{a | \exists b.(a, b) \in X\}$
- Relational product: $\{(a, c) \mid \exists b.(a, b) \in X \wedge (b, c) \in Y)\}$ ($\cap$ + existential quantification)



- Replace: bit reordering

- Operation cost proportional to # of nodes in BDD
  - To minimize cost, keep BDDs in reduced form
  - Implicitly refer to ROBDDs simply as BDDs

Bit Ordering

- Ordering of bits in BDDs is arbitrary
  - Any permutation is valid
  - Some permutations lead to smaller (reduced) BDDs

BuDDy

- Publicly available BDD package
  - Written in C
  - Supports dynamic variable reordering
  - Features node garbage collection
  - Groups bits into domains

Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions

Points-to Algorithm

- Java extension of Andersen's analysis
  - Flow-insensitive
  - Context-insensitive
  - Subset-based constraints
- All constraints generated ahead of time to separate constraint generation from solver
  - Call graph for constraint generation obtained using CHA

Points-to Algorithm

- 4 types of statements
  - Allocation: $a : l := new\, C$
  - Simple assignment: $l_2 := l_1$
  - Field store: $q.f := l$
  - Field load: $l := p.f$
- 2 relations
  - Points-to: pt
    - $pt(l)$ denotes the set of objects that l may point to
  - Assignment-edge: $\rightarrow$
    - $a \rightarrow b$ indicates that b may point to any object that a may point to

Inference Rules

- Simple assignments

$$\frac{l_1 \longrightarrow l_2 \quad o \in pt(l_1)}{o \in pt(l_2)}$$

- Field stores

$$\frac{o_2 \in pt(l) \quad l \longrightarrow q.f \quad o_1 \in pt(q)}{o_2 \in pt(o_1.f)}$$

- Field loads

$$\frac{p.f \longrightarrow l \quad o_1 \in pt(p) \quad o_2 \in pt(o_1.f)}{o_2 \in pt(l)}$$

PTA Solver Algorithm

```
init
repeat
    repeat
        Process simple assignments
    until no change
    Process field stores
    Process field loads
until no change
```

BDD Implementation

Recall:

```
X: O a = new O();
Y: O b = new O();
Z: O c = new O();
      a = b;
      b = a;
      c = b;
```

Points-to set: { (a,X) (b,Y) (c,Z) (a,Y) (b,X) (c,X) (c,Y) }

Encoding the Example Points-to Set as a BDD

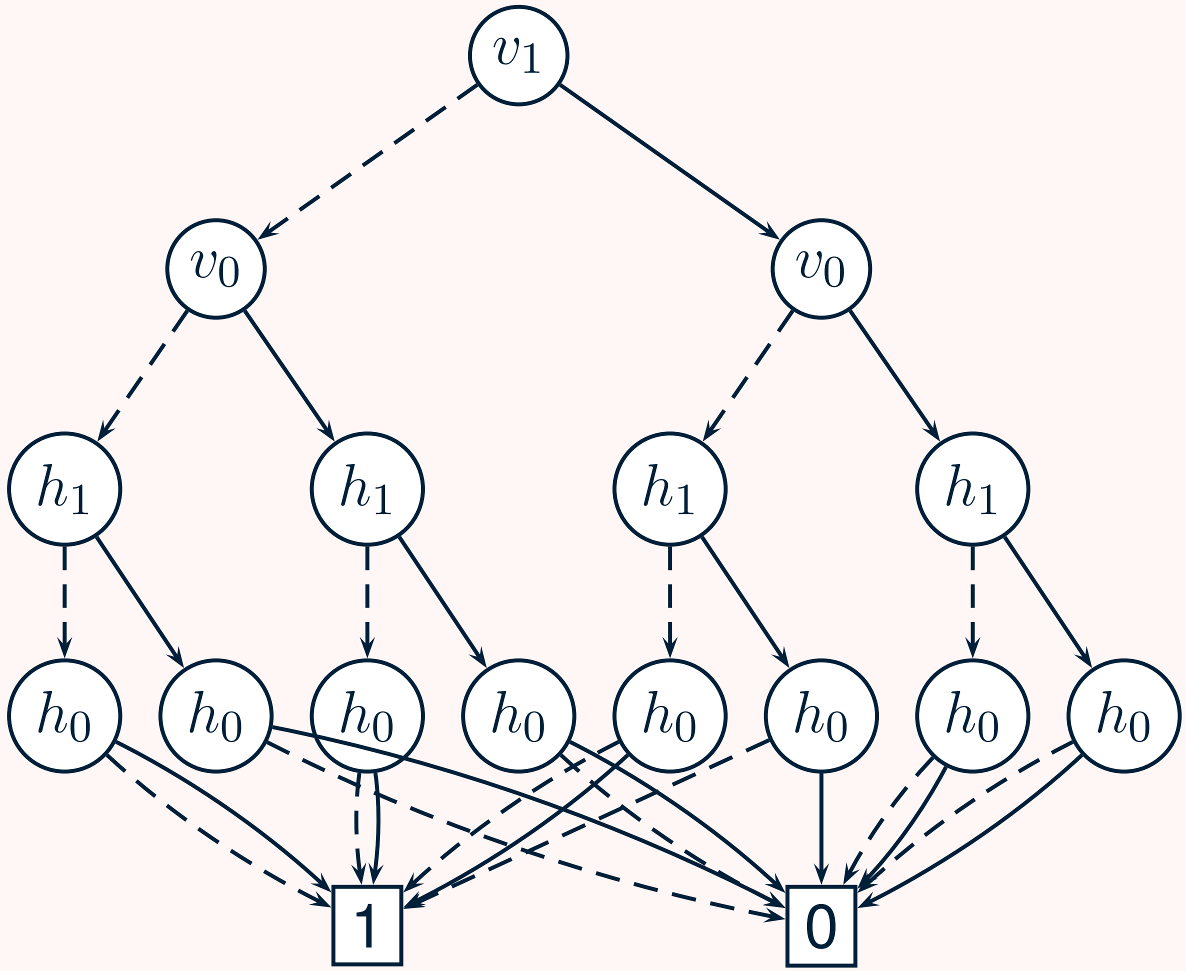
- Points-to set contains pairs of the form (v, h) where v is a variable and h is a heap location.
- Need two domains:
  - $V = \{a, b, c\}$
  - $H = \{X, Y, Z\}$
- Points-to set $P \subseteq V \times H$
- Need $\lceil log_2(|V|) \rceil = 2$ bits for each element of V
  - Represent elements of V as binary string $v_1 v_0$

	$v_1 v_0$
a	0 0
b	0 1
c	1 0

  - Idem for H

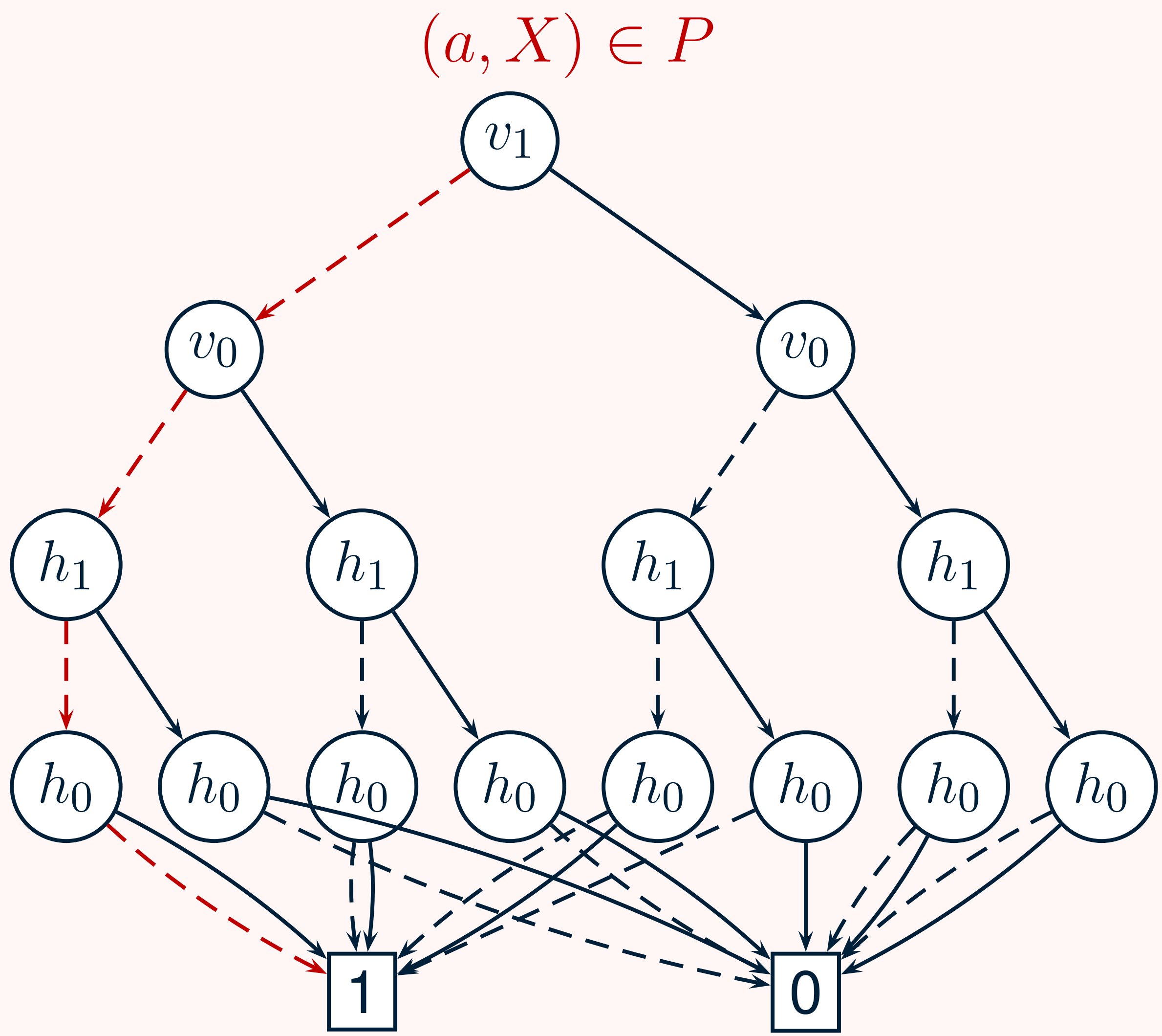
Encoding the Example Points-to Set as a BDD (2)

- $(v, h) \in P \Leftrightarrow v_1 v_0 h_1 h_0$ is mapped to $\boxed{1}$ in the BDD



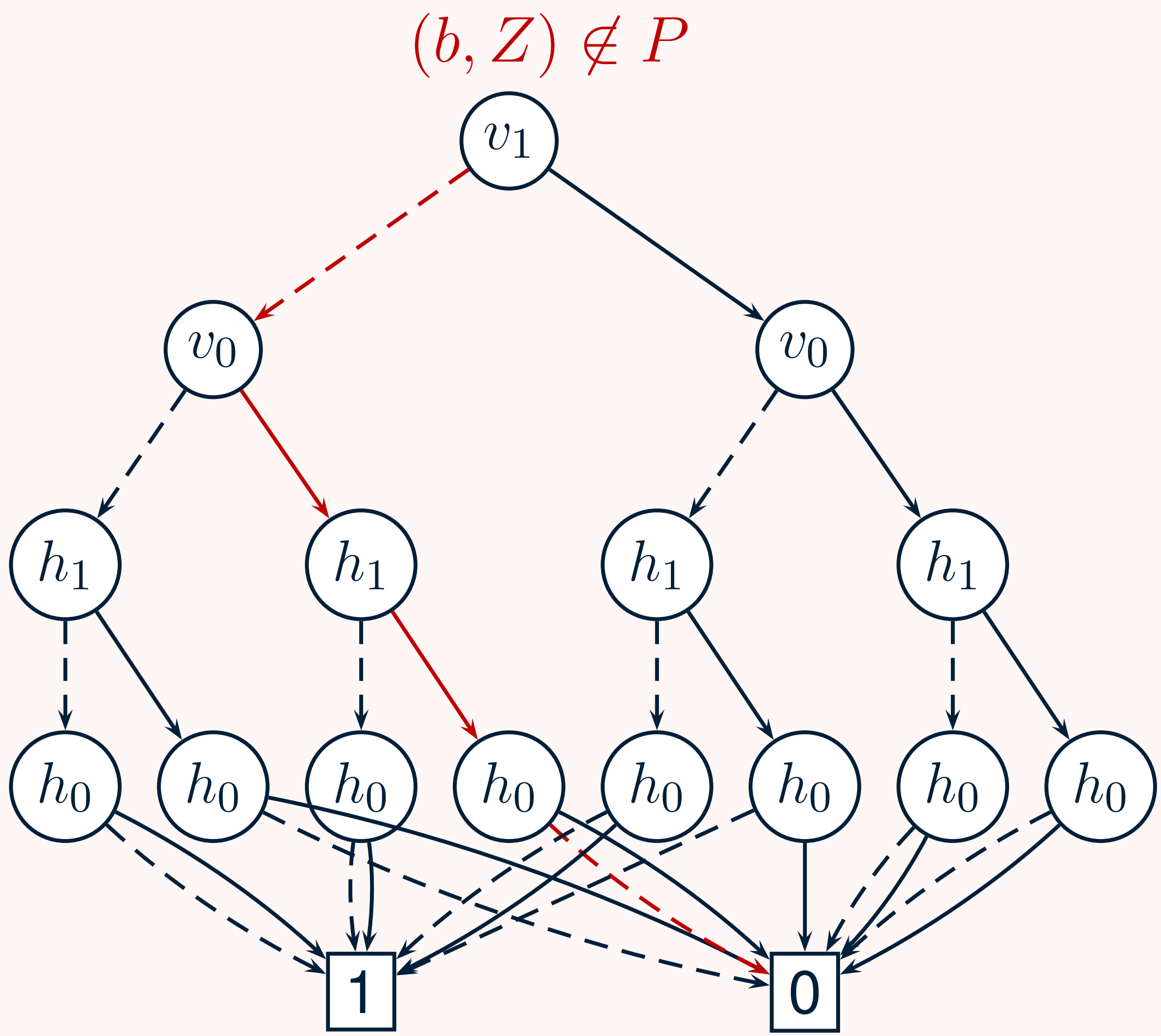
Encoding the Example Points-to Set as a BDD (2)

- $(v, h) \in P \Leftrightarrow v_1 v_0 h_1 h_0$ is mapped to $\boxed{1}$ in the BDD

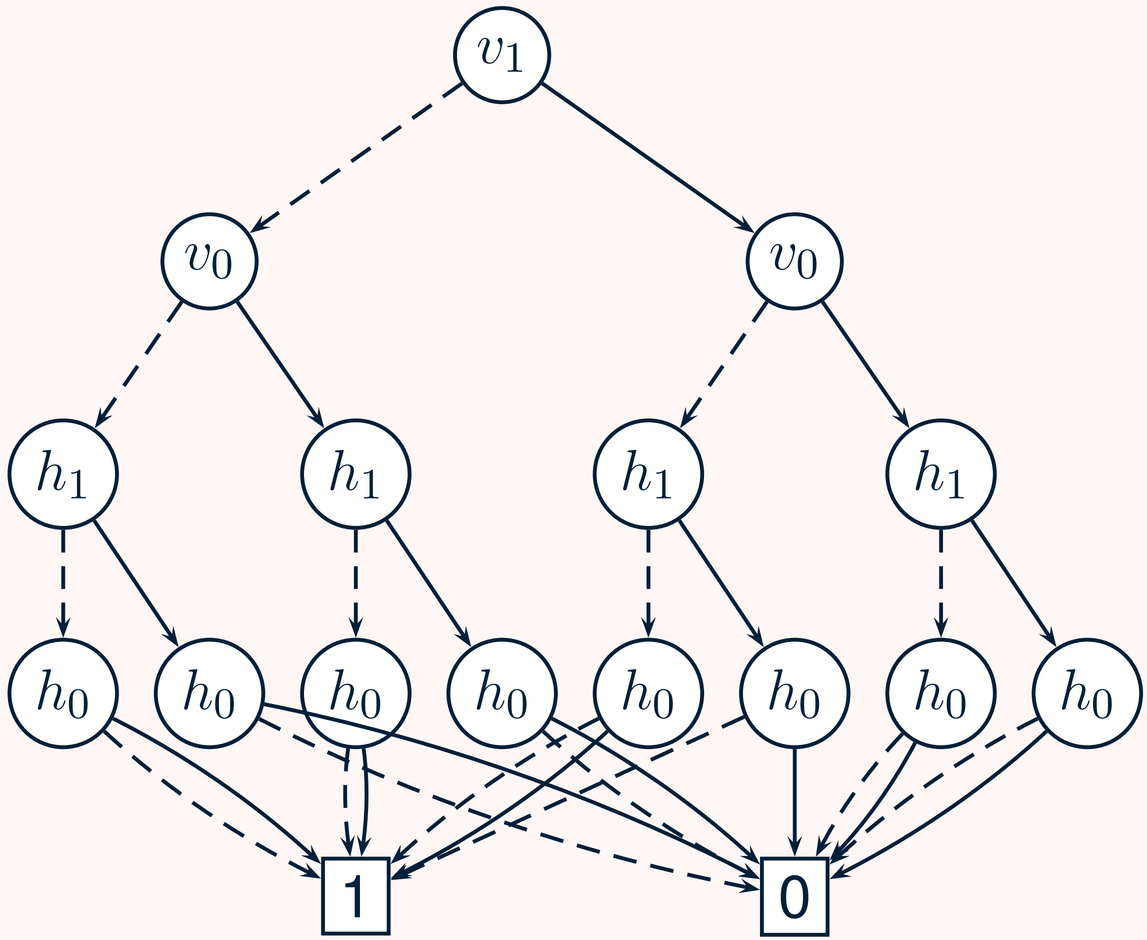


Encoding the Example Points-to Set as a BDD (2)

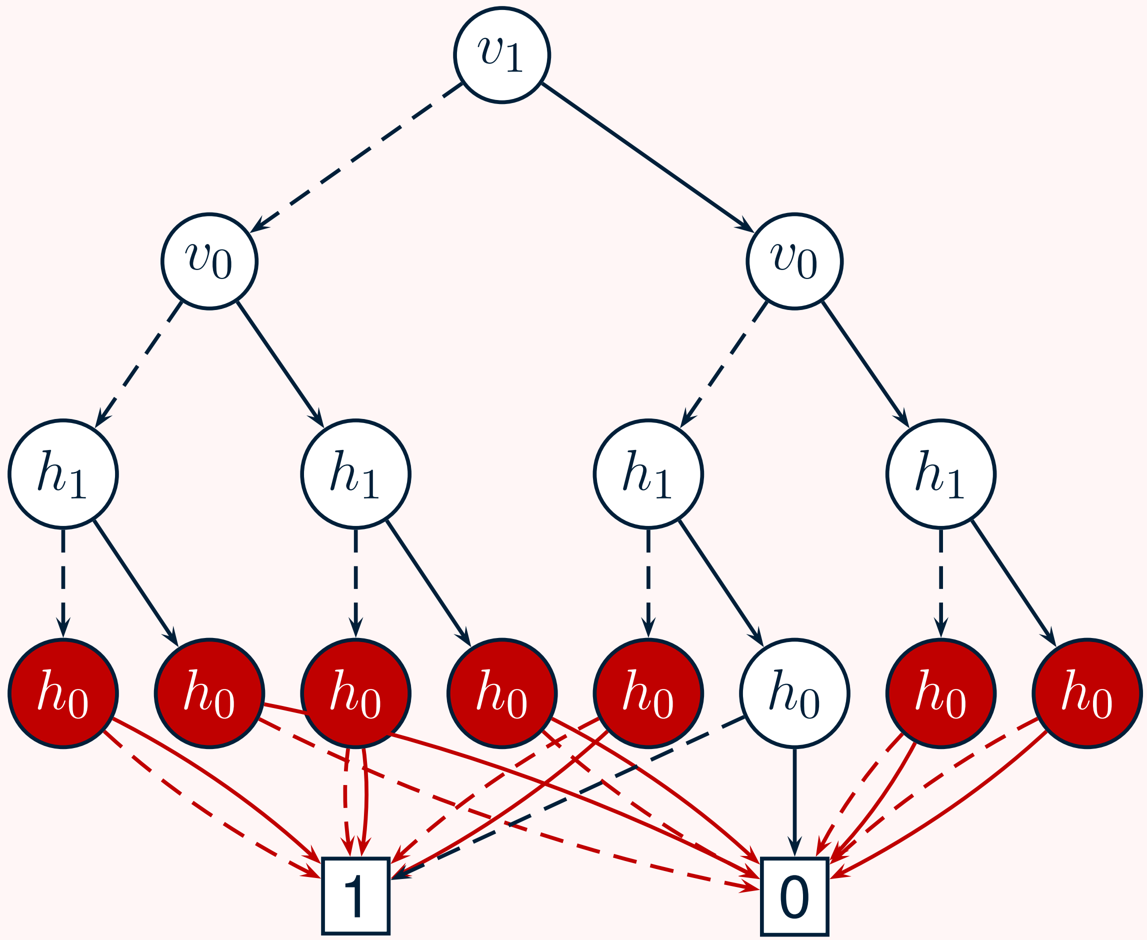
- $(v, h) \in P \Leftrightarrow v_1 v_0 h_1 h_0$ is mapped to $\boxed{1}$ in the BDD



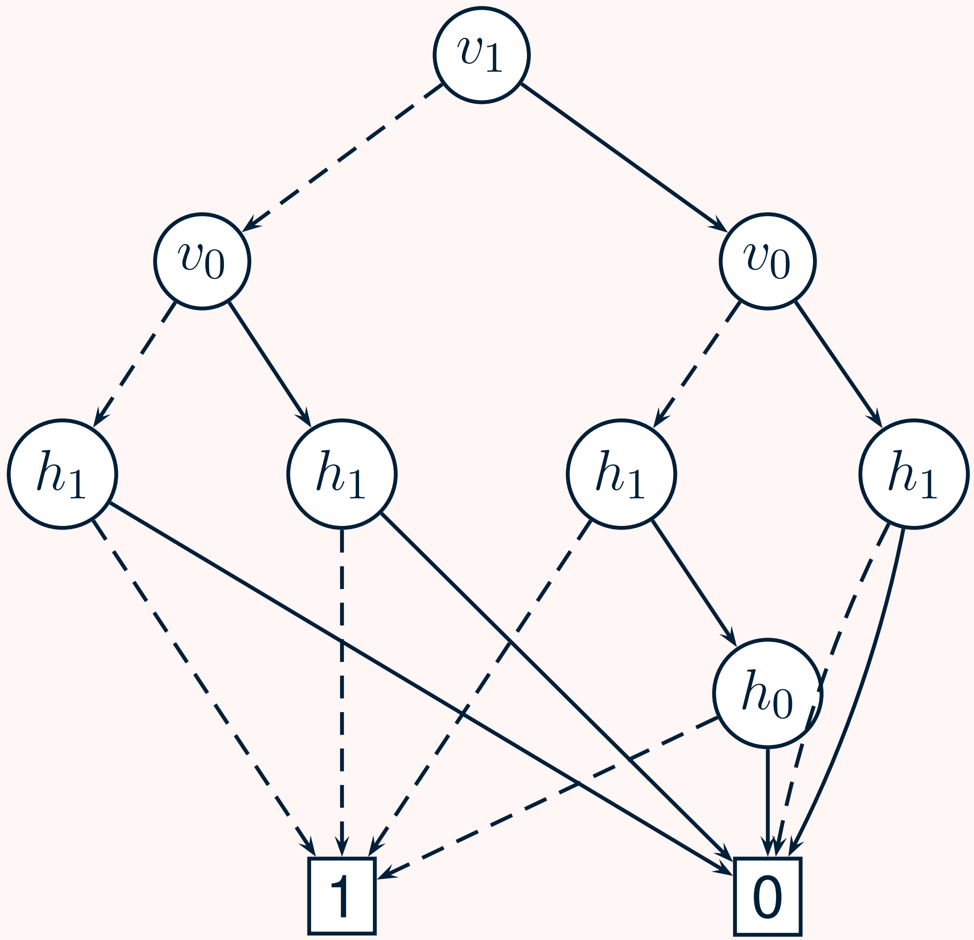
BDD Representation



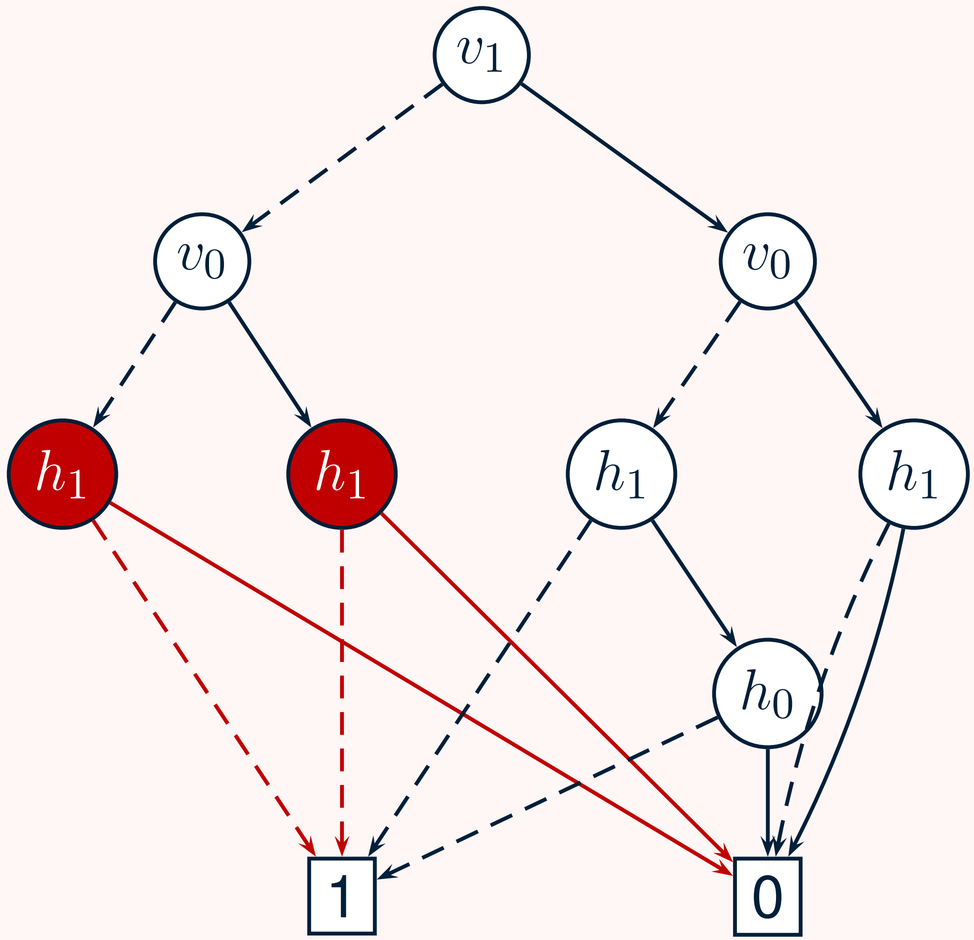
BDD Reduction (1)



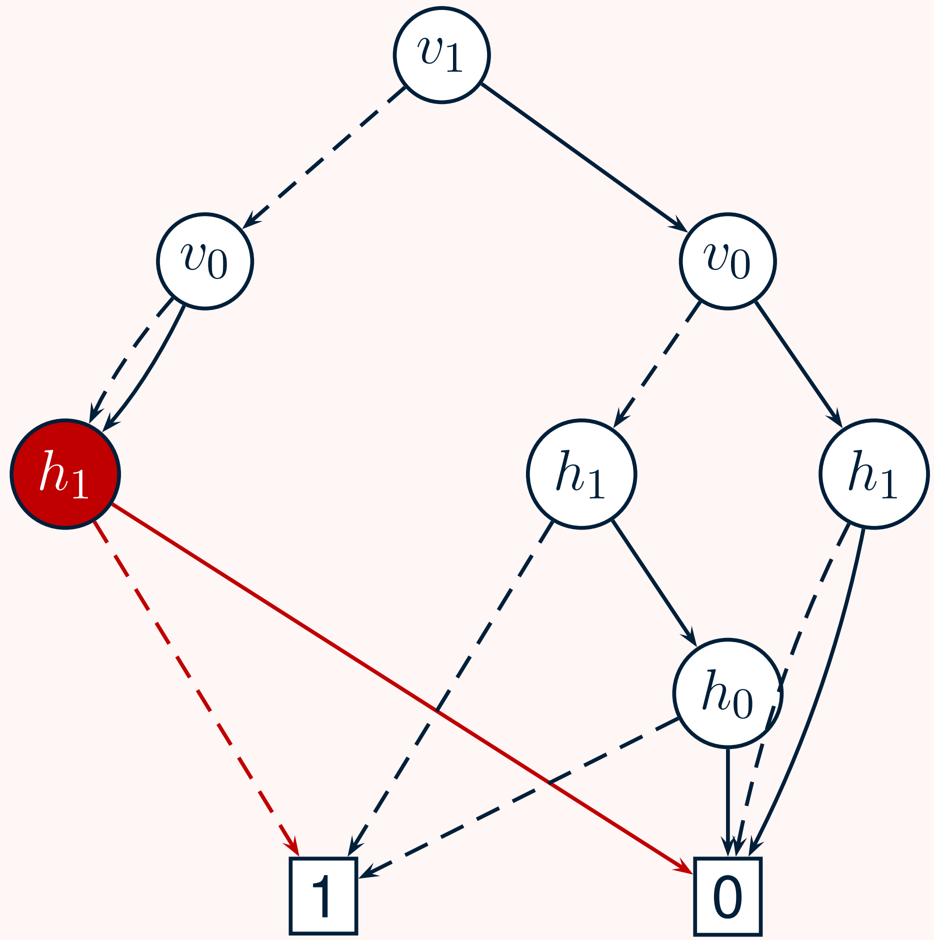
BDD Reduction (2)



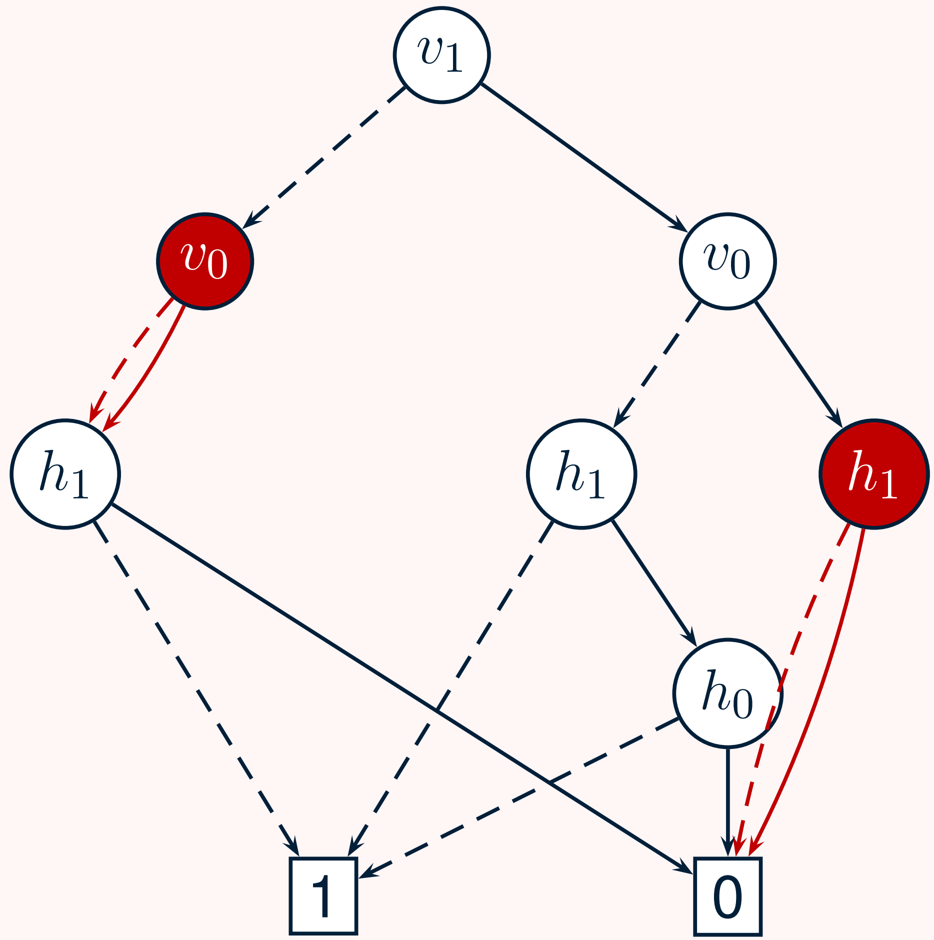
BDD Reduction (3)



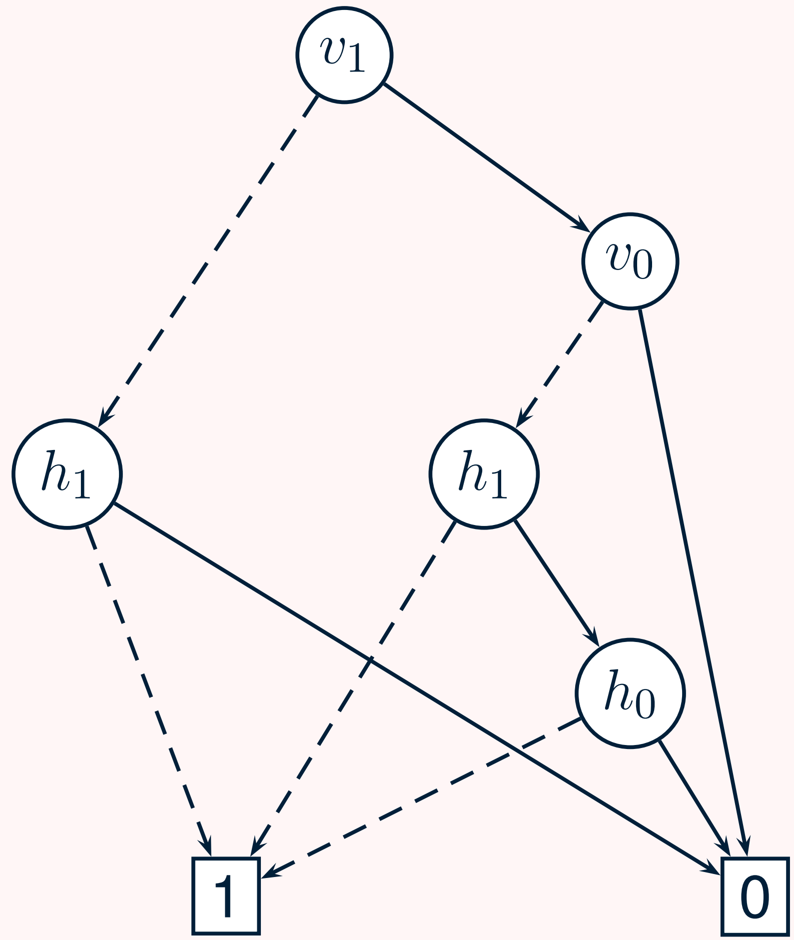
BDD Reduction (4)



BDD Reduction (5)



Reduced BDD Representation



General BDD Implementation

- Need 5 domains
  - $V1$, $V2$: Reference variables
    - Need two domains to represent pairs in $V \times V$
  - $H1, H2$: Allocation sites
    - Need two domains to represent the points-to set of object fields
  - FD: field signatures

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	
H1	X Y Z		

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b,Y) (c,Z)

(b $\longrightarrow$ a) (a $\longrightarrow$ b) (b $\longrightarrow$ c)

relprod

Domains	Points-to			Edges			New points-to
V1	a	b	c	b	a	b	
V2				a	b	c	
H1	X	Y	Z				

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \rightarrow a)$

(b,Y) $(a \rightarrow b)$ relprod

(c,Z) $(b \rightarrow c)$

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

relprod

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \rightarrow a)$
(b,Y) $(a \rightarrow b)$
(c,Z) $(b \rightarrow c)$

relprod

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $\quad$ $(b \longrightarrow a)$
(b,Y) $\quad$ $(a \longrightarrow b)$ $\qquad$ relprod
(c,Z) $\quad$ $(b \longrightarrow c)$

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b a c
H1	X Y Z		X Y Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b			
V2				a	b	c	b	a	c
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X)	$(b \longrightarrow a)$
(b,Y)	$(a \longrightarrow b)$
(c,Z)	$(b \longrightarrow c)$

replace

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b a c
H1	X Y Z		X Y Y

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b $\longrightarrow$ a)
(b,Y) (a $\longrightarrow$ b)
(c,Z) (b $\longrightarrow$ c)

replace

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b	b	a	c
V2				a	b	c			
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b	b	a	c
V2				a	b	c			
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $\quad$ (b $\longrightarrow$ a)
(b,Y) $\quad$ (a $\longrightarrow$ b)
(c,Z) $\quad$ (b $\longrightarrow$ c)

union

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	b a c
V2		a b c	
H1	X Y Z		X Y Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b) union
(c,Z) (b $\rightarrow$ c)

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

(b $\rightarrow$ a)
(a $\rightarrow$ b)
(b $\rightarrow$ c)

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X)
(b,Y)
(c,Z)

($b \rightarrow a$)
($a \rightarrow b$)
($b \rightarrow c$)

relprod

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b,Y) (c,Z)

$(b \rightarrow a)$
$(a \rightarrow b)$
$(b \rightarrow c)$

relprod

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	c
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

(b $\rightarrow$ a) (a $\rightarrow$ b) (b $\rightarrow$ c)

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	c
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

(b $\longrightarrow$ a)
(a $\longrightarrow$ b)
(b $\longrightarrow$ c)

replace

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	c
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

$(b \rightarrow a)$
$(a \rightarrow b)$
$(b \rightarrow c)$

replace

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	c
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \rightarrow a)$
(b,Y) $(a \rightarrow b)$
(c,Z) $(b \rightarrow c)$

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	c
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

$(b \rightarrow a)$
$(a \rightarrow b)$
$(b \rightarrow c)$

union

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	c
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				X

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b) union
(c,Z) (b $\rightarrow$ c)

Domains	Points-to							Edges			New
V1	a	b	c	b	a	c	c	b	a	b	
V2								a	b	c	
H1	X	Y	Z	X	Y	Y	X				

Important Relations

- $pointsTo \subseteq V1 \times H1$
  points-to relation for variables
  (l points to o)
- $fieldPt \subseteq (H1 \times FD) \times H2$
  points-to relation for object fields
  ($o_1.f$ points to o_2)
- $edgeSet \subseteq V1 \times V2$
  simple assignments
  (l_2 := l_1)
- $stores \subseteq V1 \times (V2 \times FD)$
  field stores
  ($l_2.f$:= l_1)
- $loads \subseteq (V1 \times FD) \times V2$
  field loads
  (l_2 := $l_1.f$)
- $typeFilter \subseteq V1 \times H1$

Simple assignments ($l_2 := l_1$)

```
newPt1:   [V2xH1] =
          relprod( edgeSet: [V1xV2],
                   pointsTo:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:  [V2xH1],
                            V2ToV1 );
newPt3:   [V1xH1] =
          isect  ( newPt2: [V1xH1],
                   typeFilter: [V1xH1] );
pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPt3:  [V1xH1] );
```

Field stores ($q.f$:= l)

```
tmpRel1:[(V2xFD)xH1] =
    relprod( stores:   [V1x(V2xFD)],
             pointsTo:[V1xH1],
                      V1 );
tmpRel2:[(V1xFD)xH2] =
    replace( tmpRel1: [(V2xFD)xH1],
                      V2ToV1 & H1ToH2);
fieldPt:[(H1xFD)xH2] =
    relprod( tmpRel2: [(V1xFD)xH2],
             pointsTo:[V1xH1],
                      V1 );
```

Field loads (l := $p.f$)

```
tmpRel3:      [(H1xFD)xV2] =
    relprod( loads:    [(V1xFD)xV2],
             pointsTo:[V1xH1],
                       V1 );
newPt4:       [V2xH2] =
    relprod( tmpRel3: [(H1xFD)xV2],
             fieldPt: [(H1xFD)xH2],
                       H1xFD );
newPt5:       [V1xH1] =
    replace( newPt4:  [V2xH2],
                       V2ToV1 & H2ToH1);
```

Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions

Experimental Setup

- Subset-based constraints generated by SPARK for a field-sensitive analysis
- Call graph constructed using CHA
- Effect of native methods considered (inherited from SOOT)
- 2 kinds of sets of constraints
  - Simplified (s)
  - Non-simplified (ns)
- 2 strategies for handling declared types
  - Type filtering during analysis (t)
  - Type filtering at the end of analysis (nt)

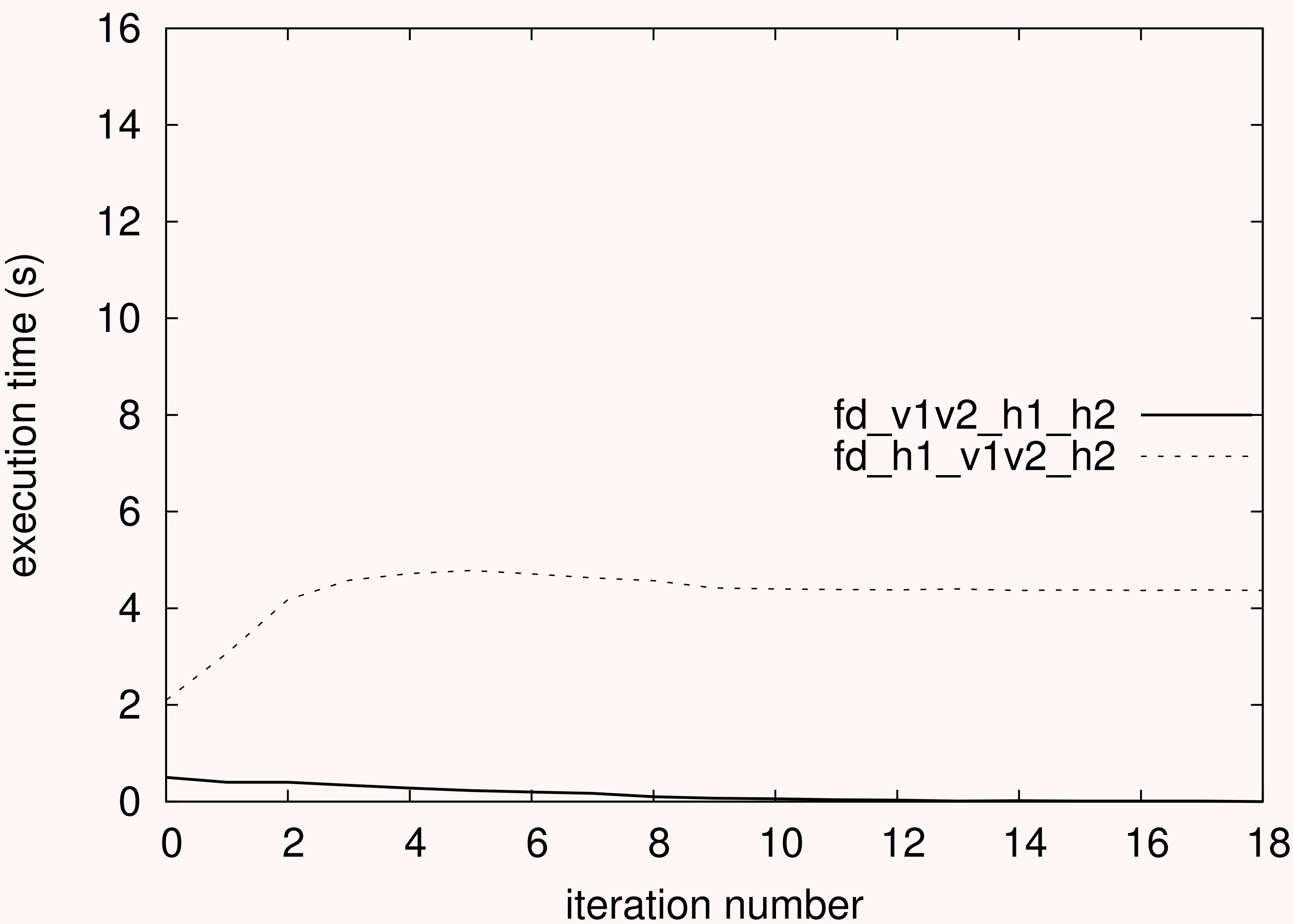
Performance Tuning – Variable Ordering

- Problem: Using the default configuration, the BDD solver cannot solve most real benchmarks.
- Profiling reveals that `relprod` operation from the inner loop is the bottleneck.
- Two factors to consider for performance
  - Relative domain ordering
  - Variable interleaving within domains

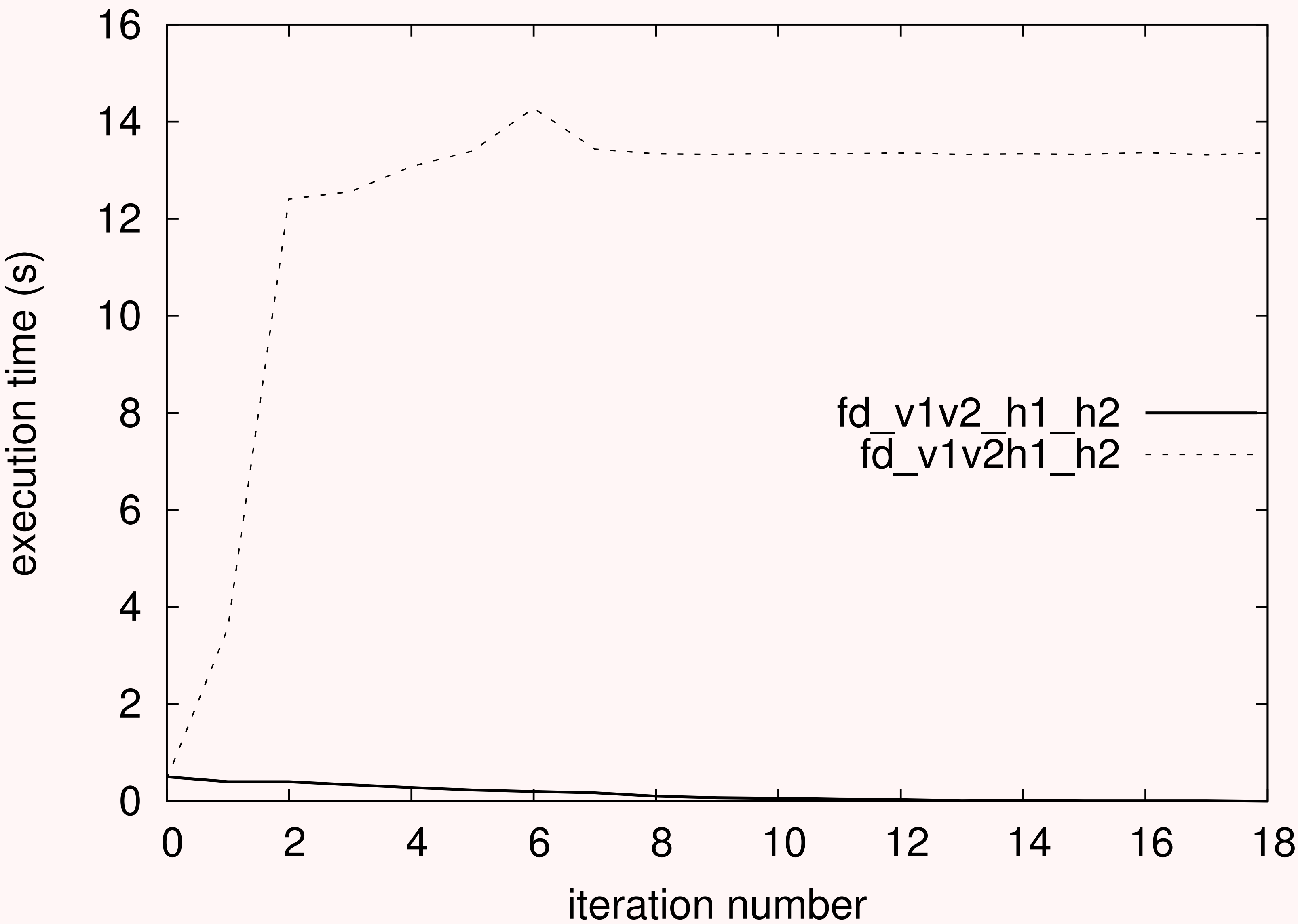
Variable Interleaving Notation

- Let FD, $V1$ be domains such that $f_0, \ldots, f_n$ are the variables of FD and $v_0, \ldots, v_n$ are the variables of domain $V1$.
  - $FDV1$ denotes the interleaving of variables, i.e. $f_0 v_0 f_1 v_1 \ldots f_n v_n$.
  - FD_V1 denotes the concatenation of variables, i.e. $f_0 f_1 \ldots f_n v_0 v_1 \ldots v_n$.
- Variables are always order from most to least significant bit to exploit unused high bits.
- `BuDDy`'s default ordering is $FDV1V2H1H2$

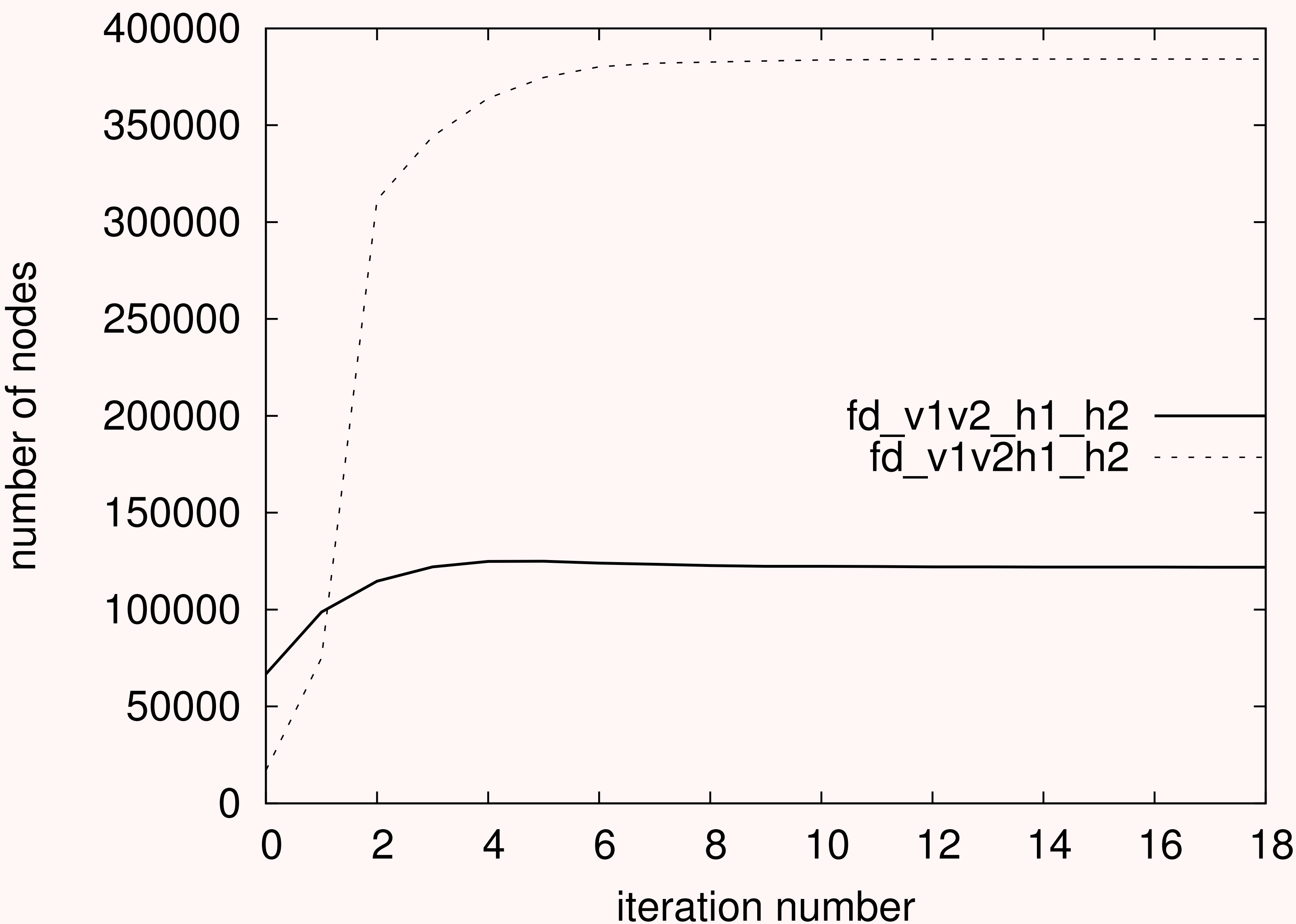
Effect of Domain Arrangement on `relprod`



Effect of Interleaving Domains on `relprod`



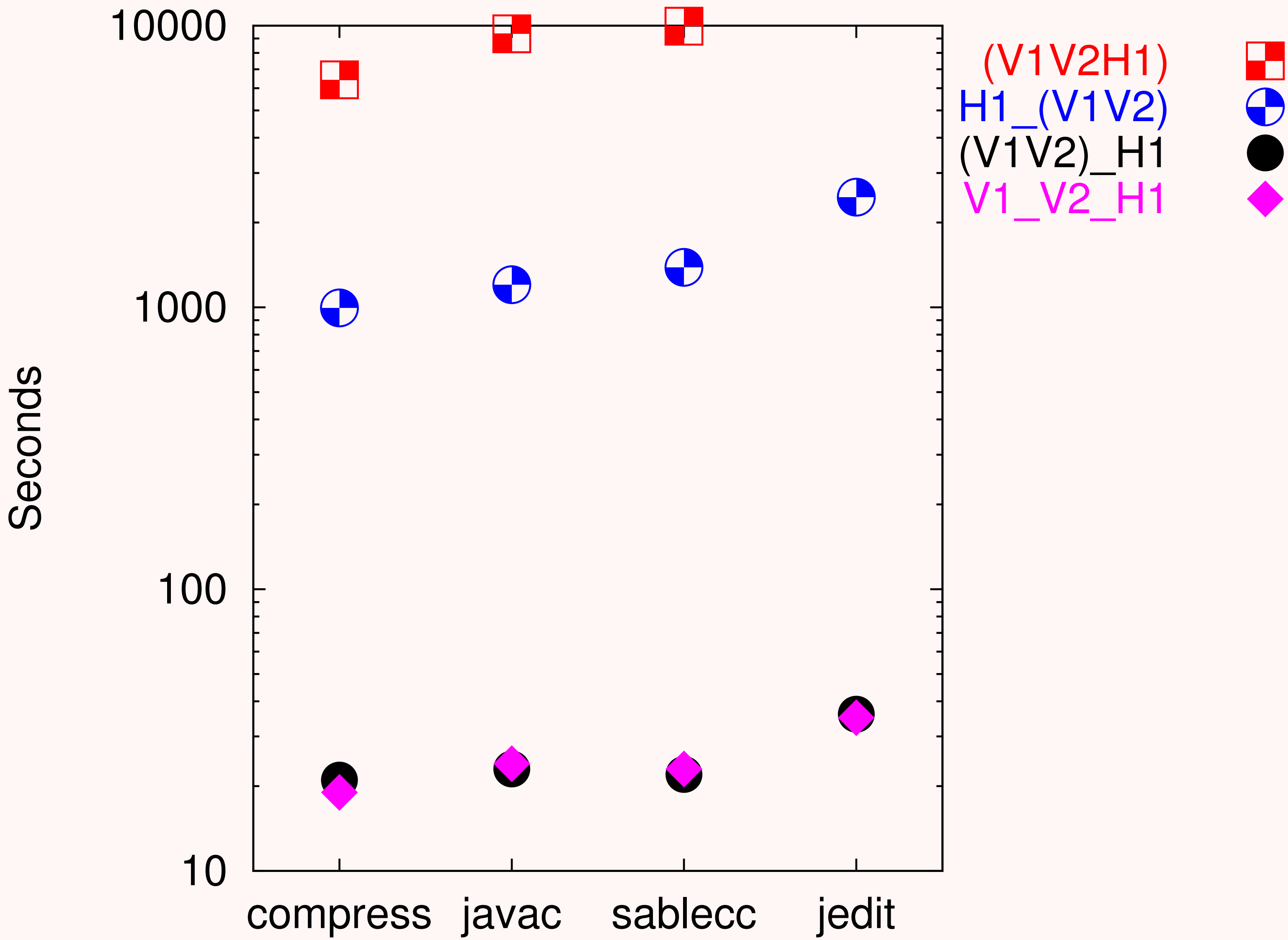
Effect of Interleaving Domains on `pointsTo`



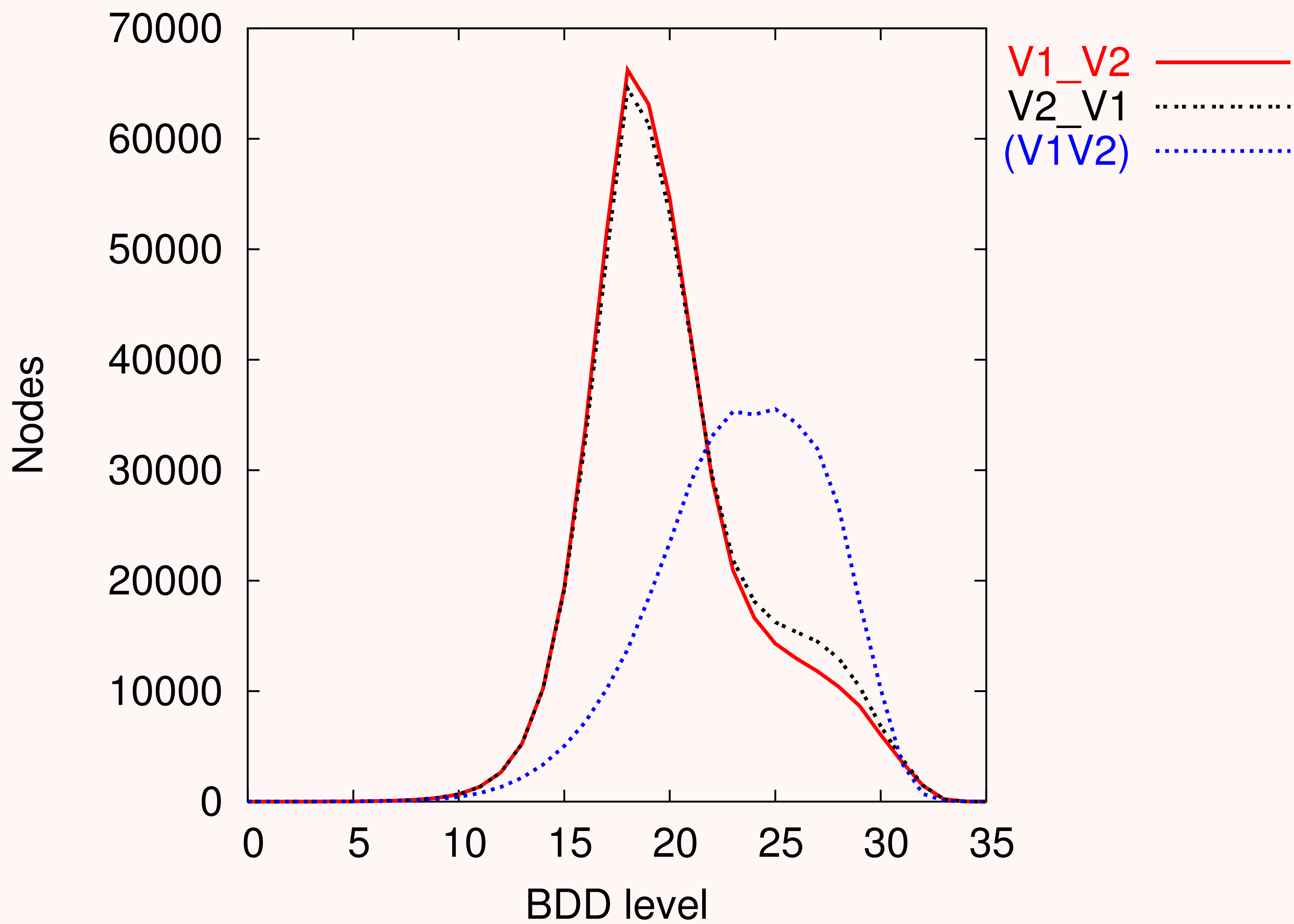
Effect of Variable Ordering on Performance

- Default ordering is good for model checking, but much too slow for PTA
- Investigate other orderings
  - Focus on the domains used in the problematic `relprod` operation, i.e. $V1, V2, H1$.
  - Reason about the impact of certain orderings on BDD size
    - Domains that feature a large amount of similarity between the sets could benefit from preventing interleaving.
    - It is easier to exploit similarities when present at the end of the variable sequence than at the beginning.

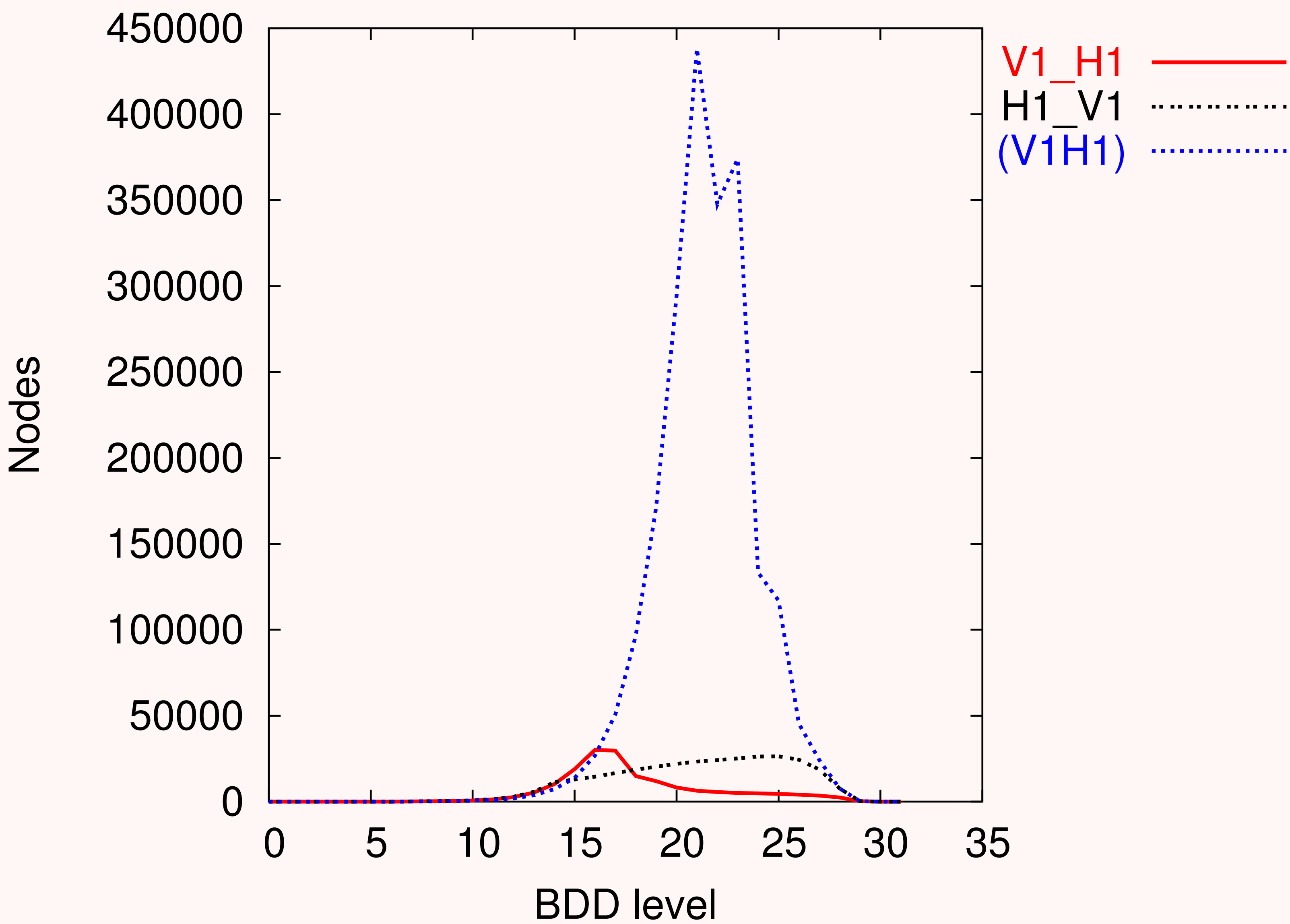
Effect of Different Orderings on Performance



Effect of Ordering on $edgeSet$



Effect of Ordering on $pointsTo$



Performance Tuning – Incrementalization

- Observation: The `relprod` operation propagates all points-to sets along all edges at every execution.
  - Most sets have already been propagated in previous iterations
  - `relprod` executes in time proportional to the # of nodes
  - The inner `relprod` is very hot

Performance Tuning – Incrementalization

- Observation: The `relprod` operation propagates all points-to sets along all edges at every execution.
  - Most sets have already been propagated in previous iterations
  - `relprod` executes in time proportional to the # of nodes
  - The inner `relprod` is very hot
- Only propagate the new part of the points-to set
  - new `pointsTo` relation remains small
  - `relprod` executes much faster

Incremental BDD-PTA Algorithm

```
newPt1:   [V2xH1] =
          relprod( edgeSet: [V1xV2],
                   pointsTo:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:  [V2xH1],
                            V2ToV1 );



pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPt2:  [V1xH1] );
```

Performance Tuning – Incrementalization

```
newPt1:   [V2xH1] =
          relprod( edgeSet: [V1xV2],
                   newPoint:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:   [V2xH1],
                            V2ToV1 );
newPoint:[V1xH1] =
          setminus( newPt2:   [V1xH1],
                    pointsTo:[V1xH1] );
pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPoint:[V1xH1] );
```

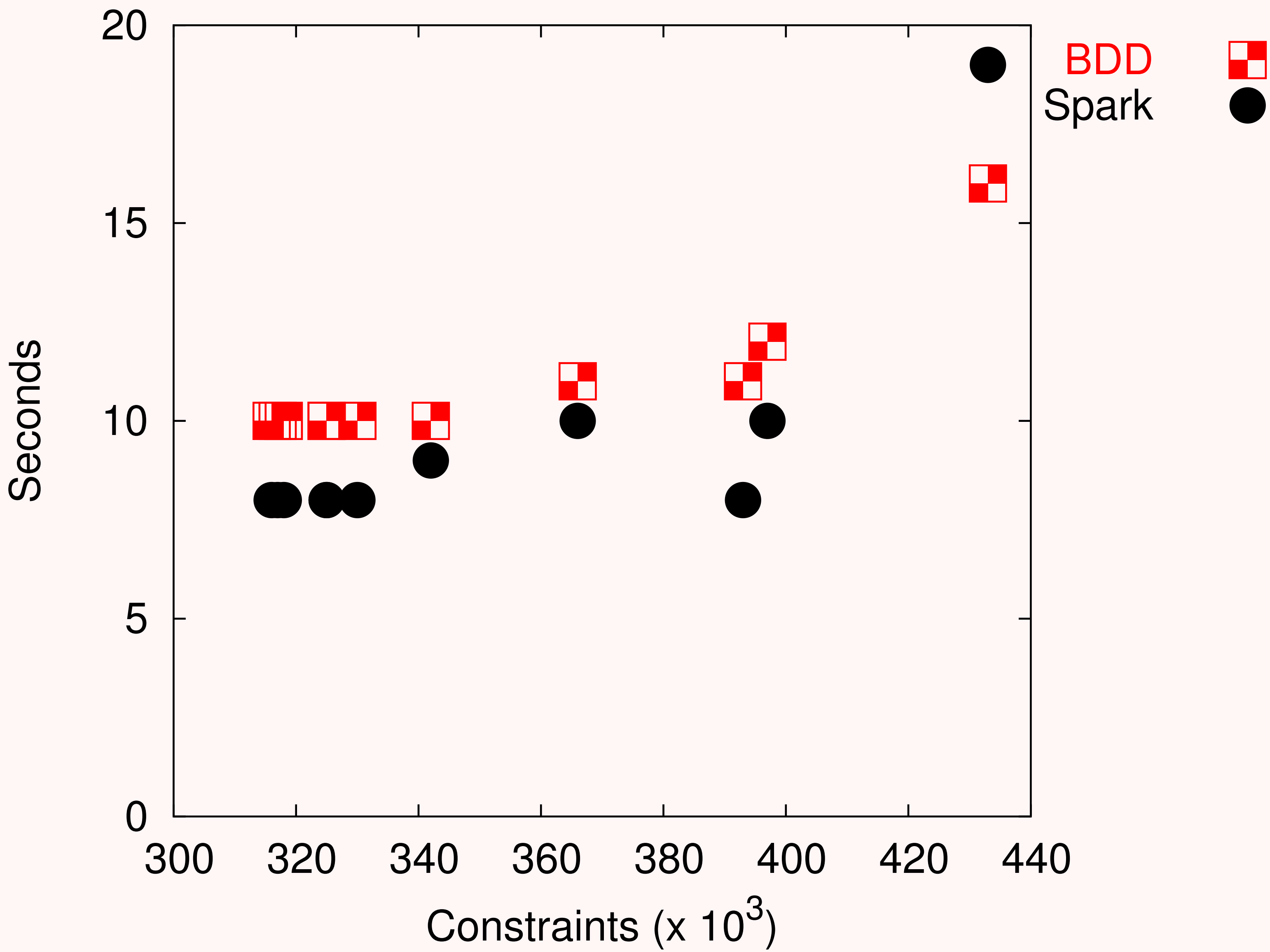
Effect of Incrementalization on Performance

benchmark	fd_V1V2_H1_2̋		FD_V1_V2_H1_H2	
	non-inc	inc	non-inc	inc
compress (s/t)	20.63	11.72	19.07	9.80
compress (ns/t)	54.46	26.83	83.63	19.66
compress (ns/nt)	145.33	71.55	228.21	58.58
javac (s/t)	22.62	14.83	23.89	10.83
javac (ns/t)	62.35	30.55	103.52	23.14
javac (ns/nt)	166.66	80.04	285.65	65.46
sablecc-j (s/t)	21.90	14.00	23.10	10.60
sablecc-j (ns/t)	63.43	30.05	110.87	22.86
sablecc-j (ns/nt)	158.33	76.53	269.30	63.82
jedit (s/t)	35.92	20.11	35.43	15.60
jedit (ns/t)	112.47	47.53	357.97	35.29
jedit (ns/nt)	336.18	150.72	783.92	120.53

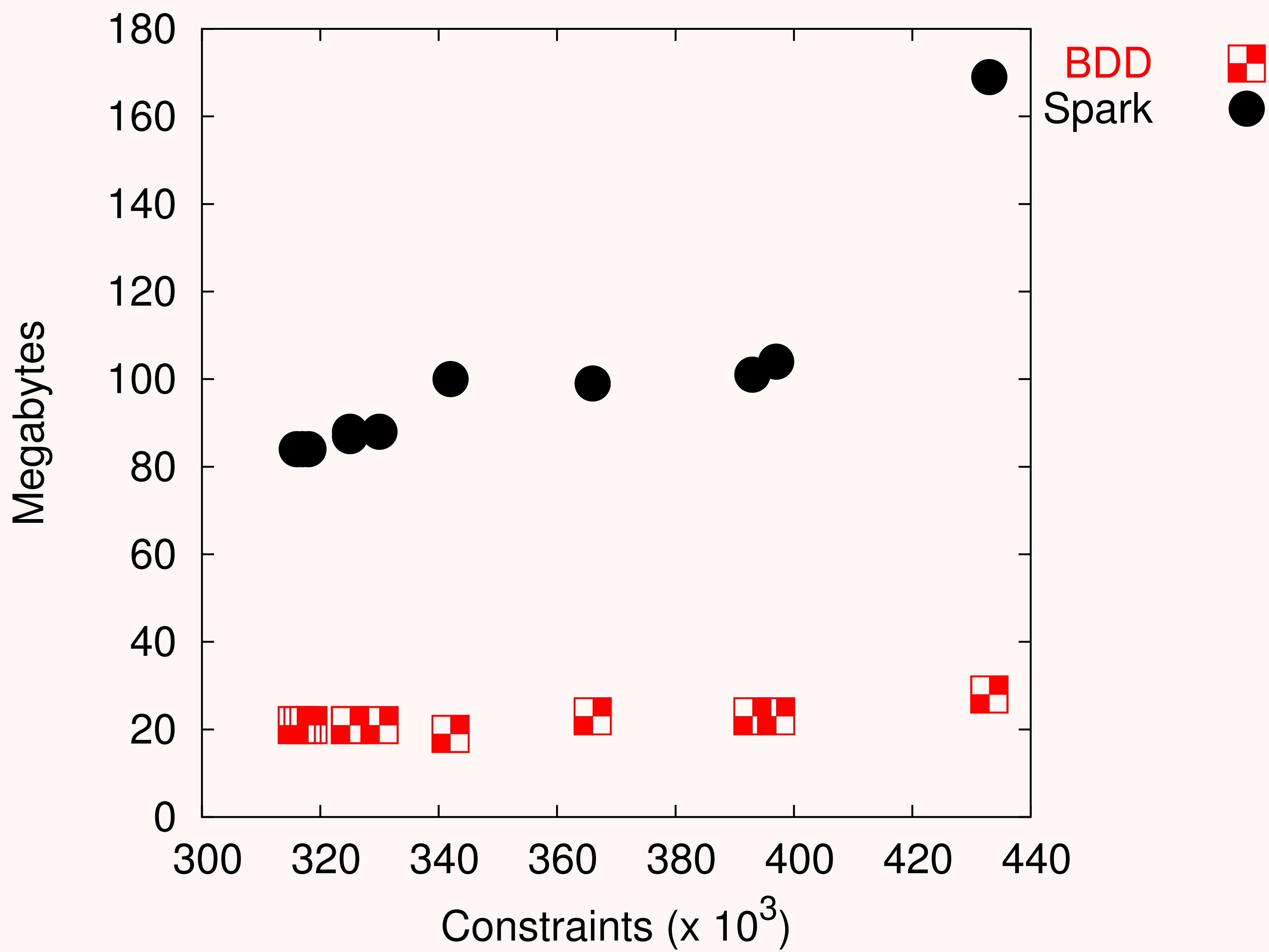
Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions

Overall Performance – Time



Overall Performance – Space



Outline

- Background
  - Points-to (reference) analysis
  - BDDs
- Points-to algorithm using BDDs
- Performance tuning
- Experimental results
- Applications
- Conclusions

Applications

- Manipulating sets makes it easy to express common problems:
  - “May/Must be aliased” analysis
  - Virtual method resolution (receiver types)
    - Inlining
    - Devirtualization
- Manipulating the solution as a BDD
  - improves performance
  - lowers development cost

Related Work

- A lot. . .
  - PTA
    - PTA algorithms
    - Efficient set representations
    - Equality-based constraints
    - . . .
  - BDDs
    - Model checking

Conclusions

- BDDs are useful in the context of PTA
- It is possible to write efficient solvers using BDD libraries “out of the box”
- Finding a good bit ordering is necessary to obtain good performance