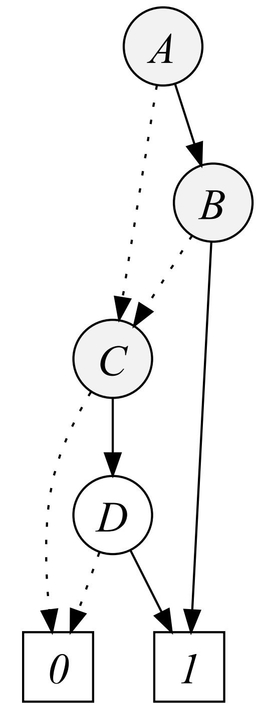
On the Role of Canonicity in Knowledge Compilation

Guy Van den Broeck and Adnan Darwiche

Jan 28, 2015, AAAI

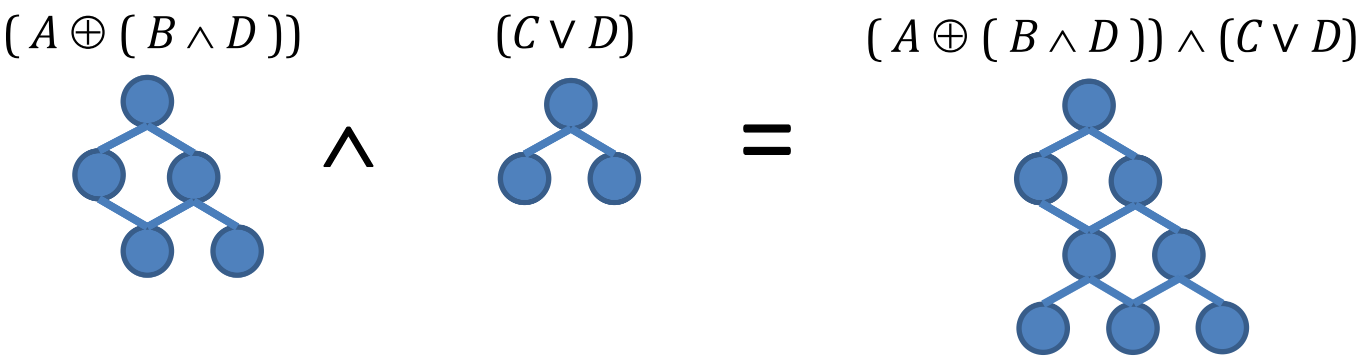
Knowledge Compilation

- Reasoning with logical knowledge bases
- Tractable languages and compilers
- Boolean circuits:
  OBDDs, d-DNNFs, SDDs, etc.
- Applications:
  - Diagnosis
  - Planning
  - Inference in probabilistic databases, graphical models, probabilistic programs
  - Learning tractable probabilistic models



Bottom-Up Compilation with Apply

- Build Boolean **combinations** of existing circuits
- Compile CNF: (1) circuit for literals (2) disjoin to get circuit for clauses (3) conjoin for CNF.
- Compile **arbitrary** sentence incrementally



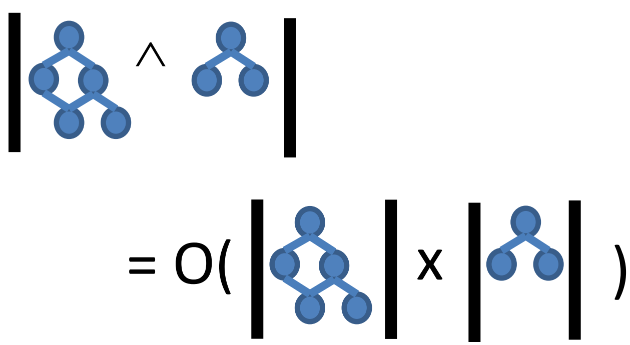
- Avoiding CNF crucial for many applications

Two Properties Under Investigation

Polytime Apply

Complexity is polynomial in size of input circuits.

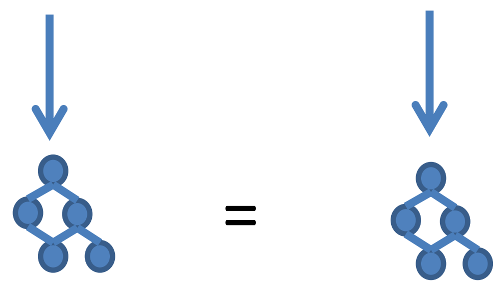
Informally: one Apply cannot blow up size.



Canonicity

Equivalent sentences have identical circuits.

$$A \wedge (C \vee D) \equiv (A \wedge C) \vee (A \wedge D)$$



What We Knew Before

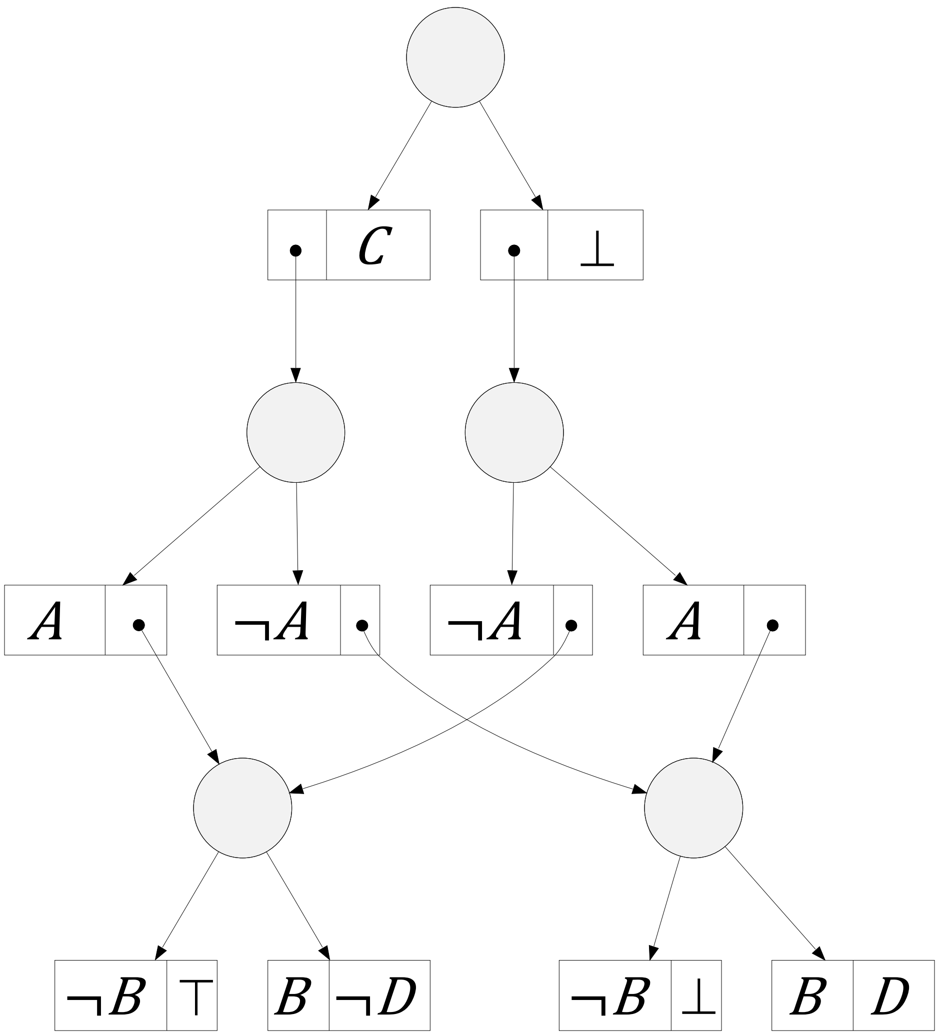
- A practical language for bottom-up compilation **requires a polytime Apply**.
  - Explains success of OBDDs
  - Why do Apply when it blows up?
  - Guided search for new languages (structured DNNF)
- Canonicity is convenient for building compilers
  - Detect/cache equivalent subcircuits

What We Knew Before

Thought We

- A practical language for bottom-up compilation **requires a polytime Apply**.
  - Explains success of OBDDs
  - Why do Apply when it blows up?
  - Guided search for new languages (structured DNNF)
- Canonicity is convenient for building compilers
  - Detect/cache equivalent subcircuits

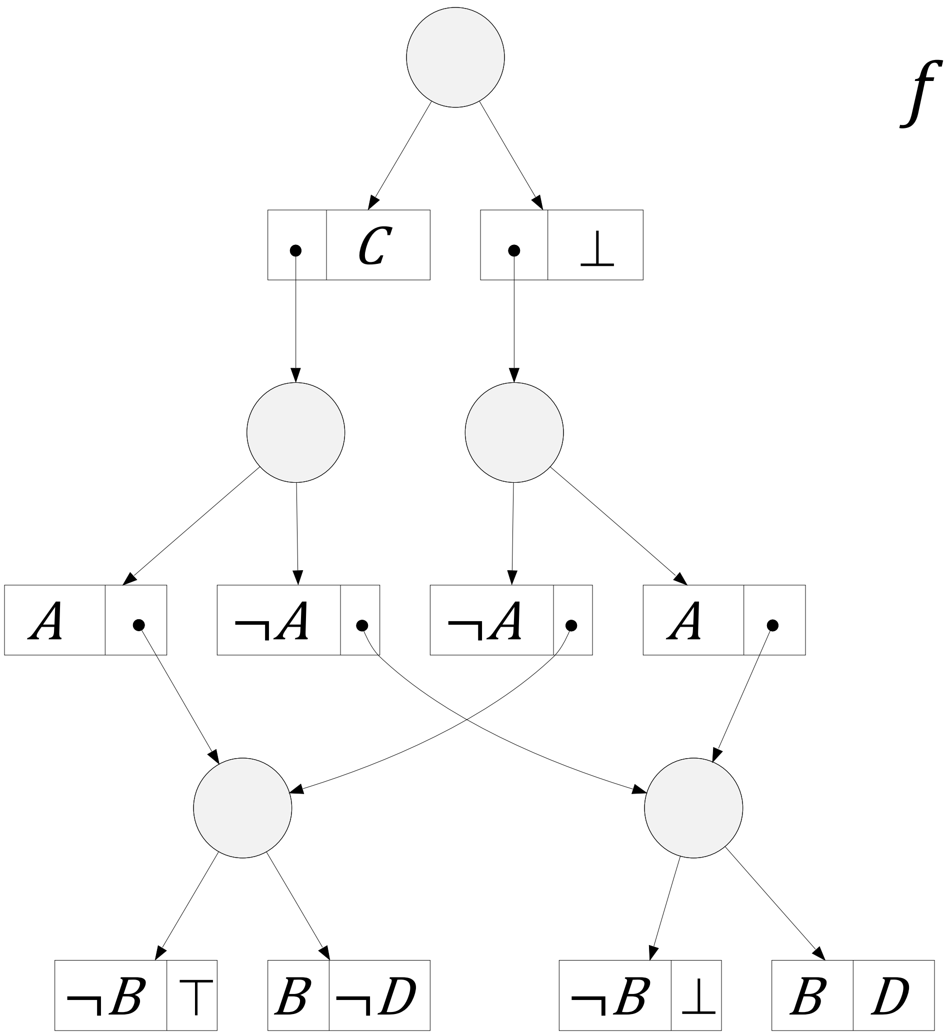
Sentential Decision Diagrams



Properties:

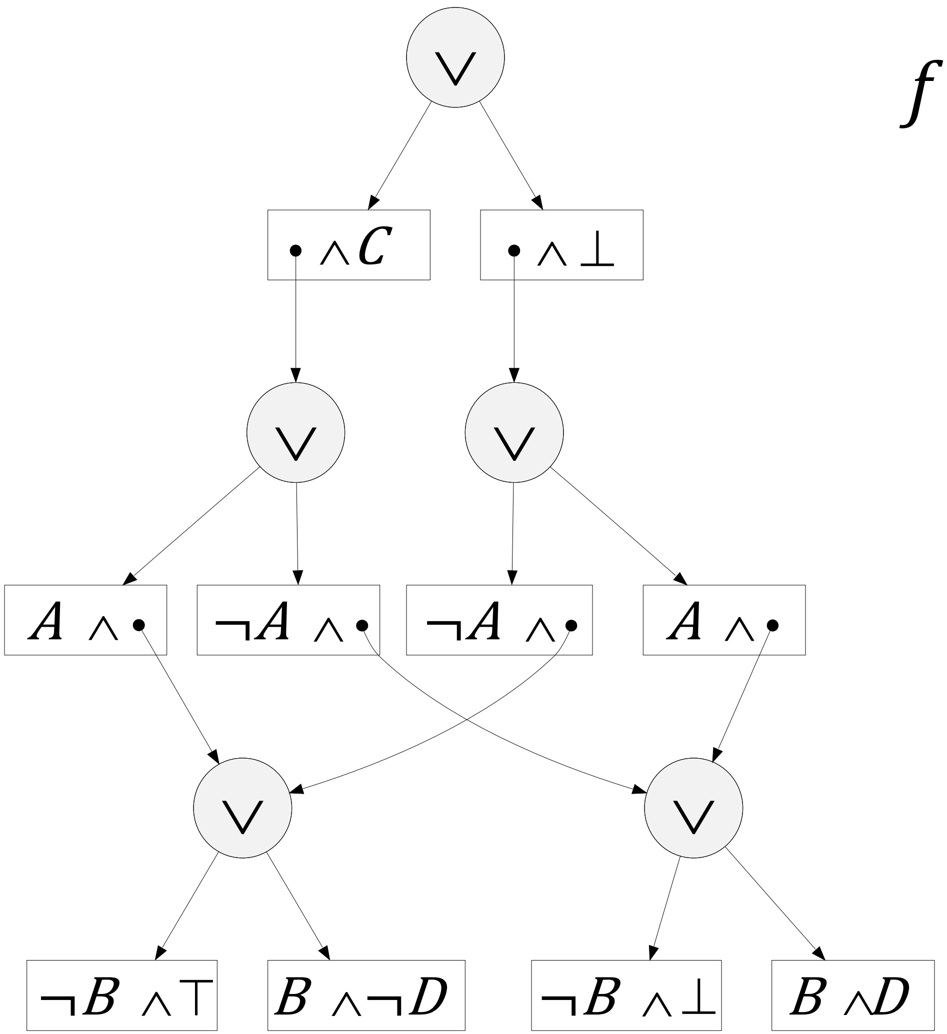
- OBDD $\subset$ SDD
- Treewidth upper bound
- Quasipolynomial separation with OBDD
- Supports OBDD queries

Sentential Decision Diagrams



$$f(A, B, C, D) = (A \oplus (B \wedge D)) \wedge C$$

Sentential Decision Diagrams

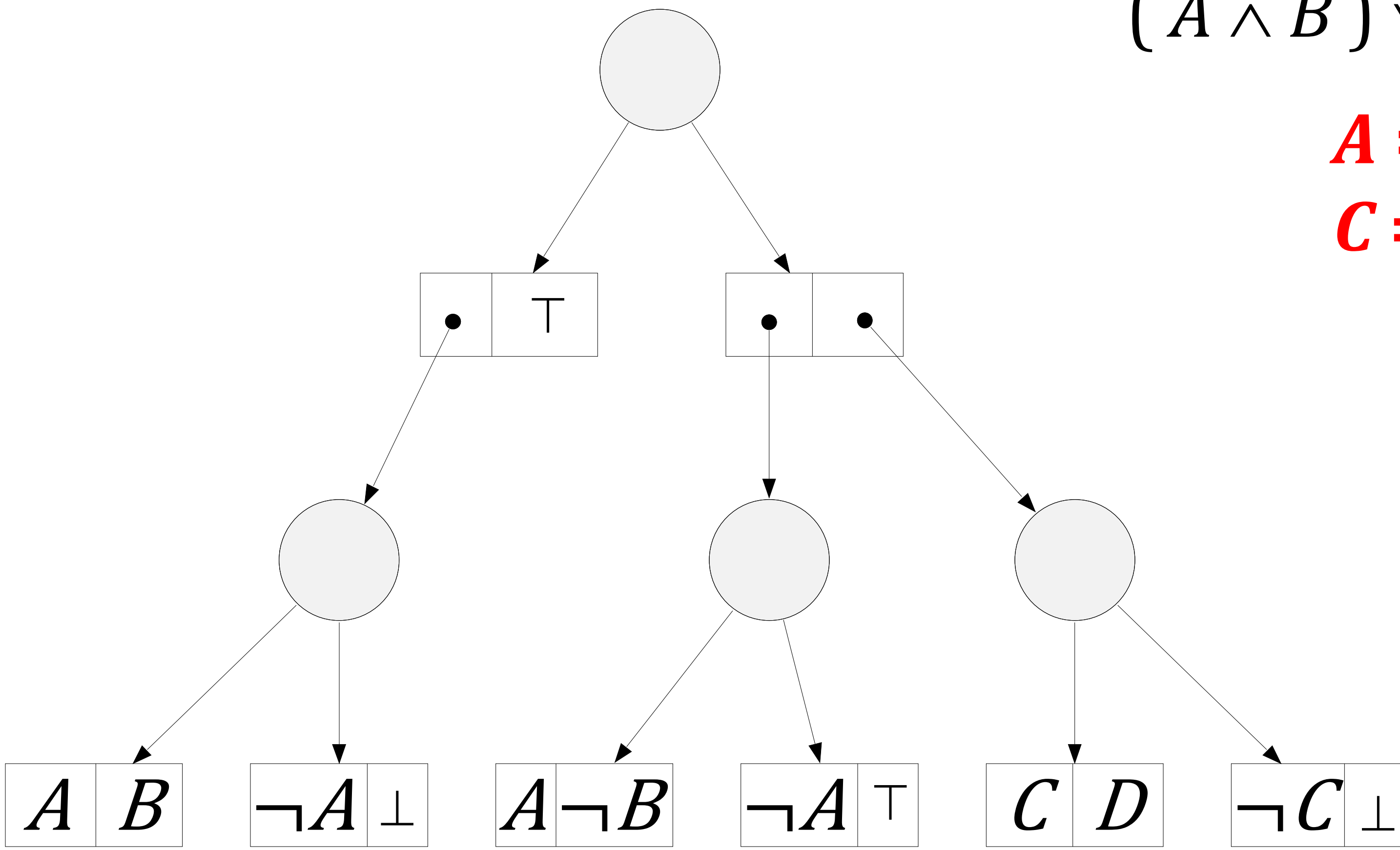


$$f(A, B, C, D) = (A \oplus (B \wedge D)) \wedge C$$

Basing Decisions on Sentences

$f(A, B, C, D) =$
$(A \wedge B) \vee (C \wedge D)$

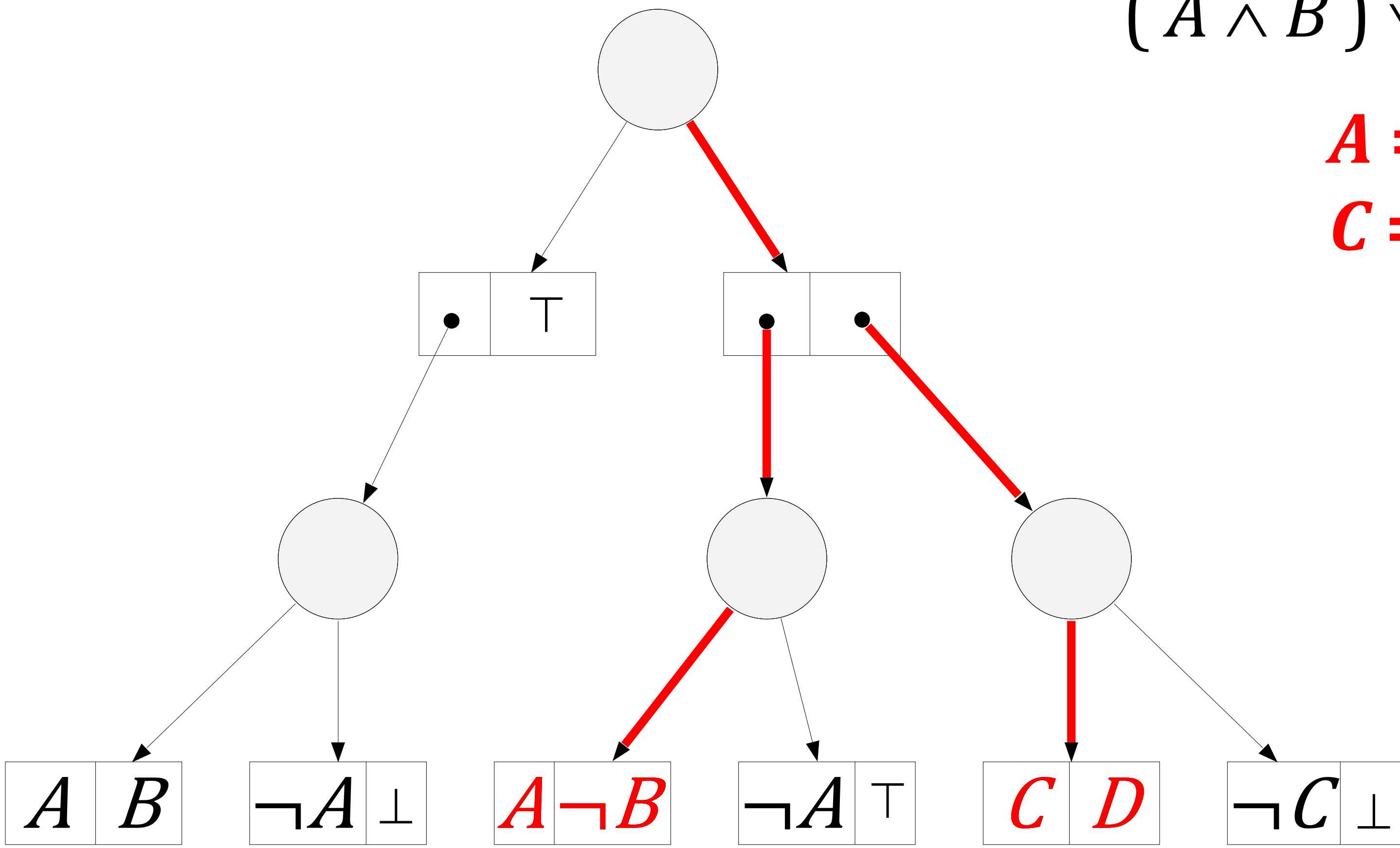
$\boldsymbol{A = t, B = f,}$
$\boldsymbol{C = t, D = t}$



Basing Decisions on Sentences

$f(A, B, C, D) =$
$(A \wedge B) \vee (C \wedge D)$

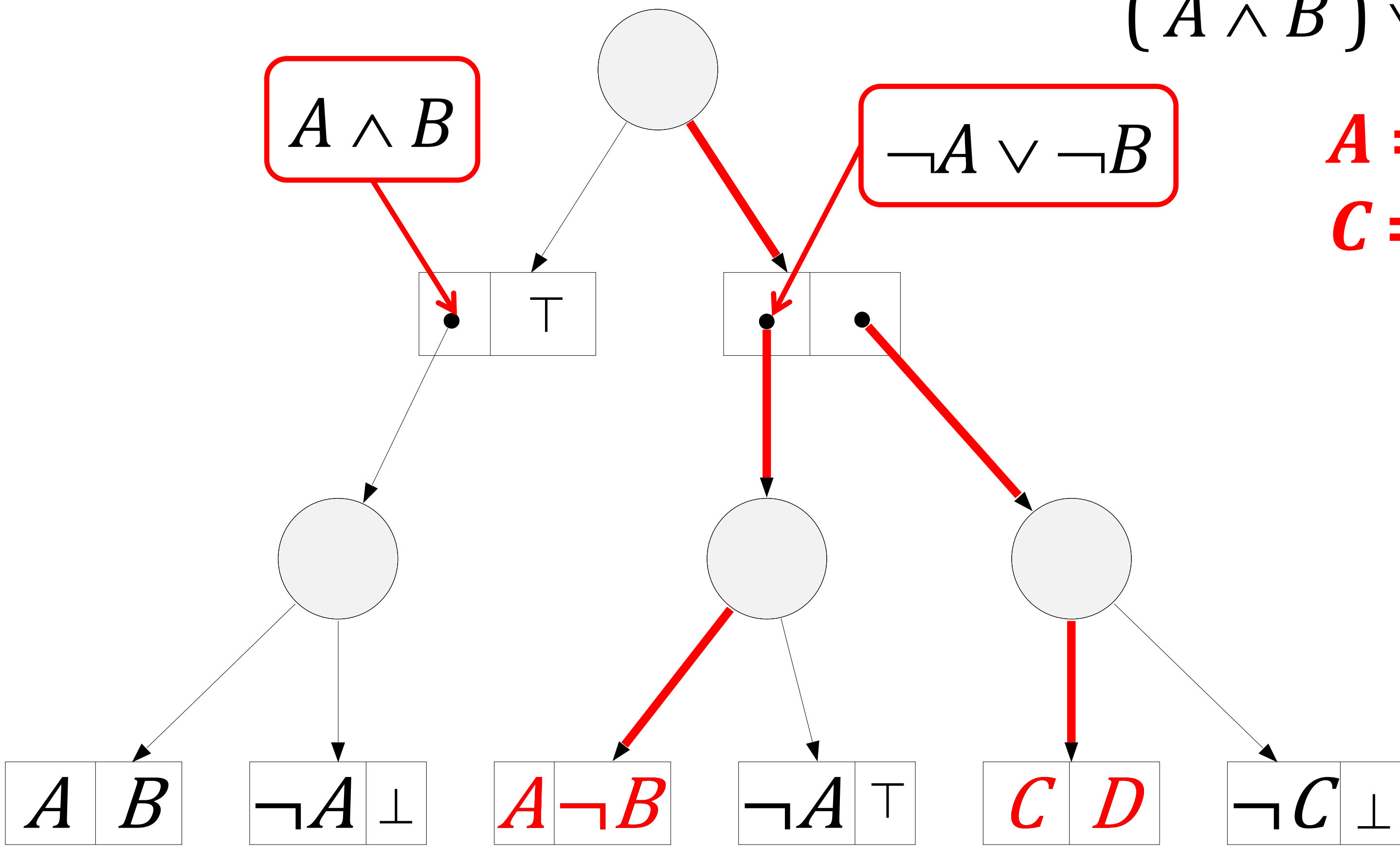
$A = t$, $B = f$,
$C = t$, $D = t$



Basing Decisions on Sentences

$f(A, B, C, D) =$
$(A \wedge B) \vee (C \wedge D)$

$A = t, B = f,$
$C = t, D = t$

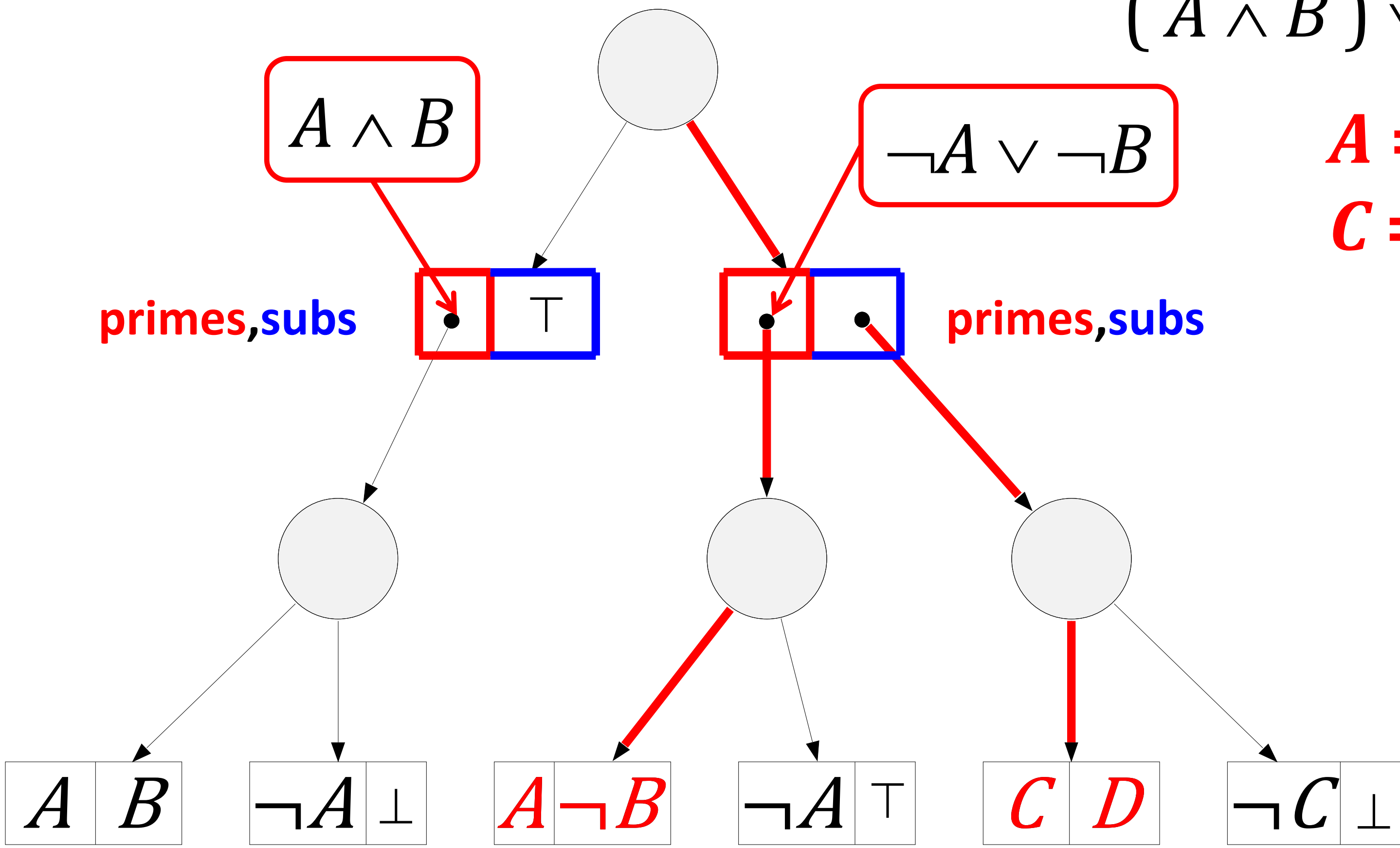


Basing Decisions on Sentences

$f(A, B, C, D) =$

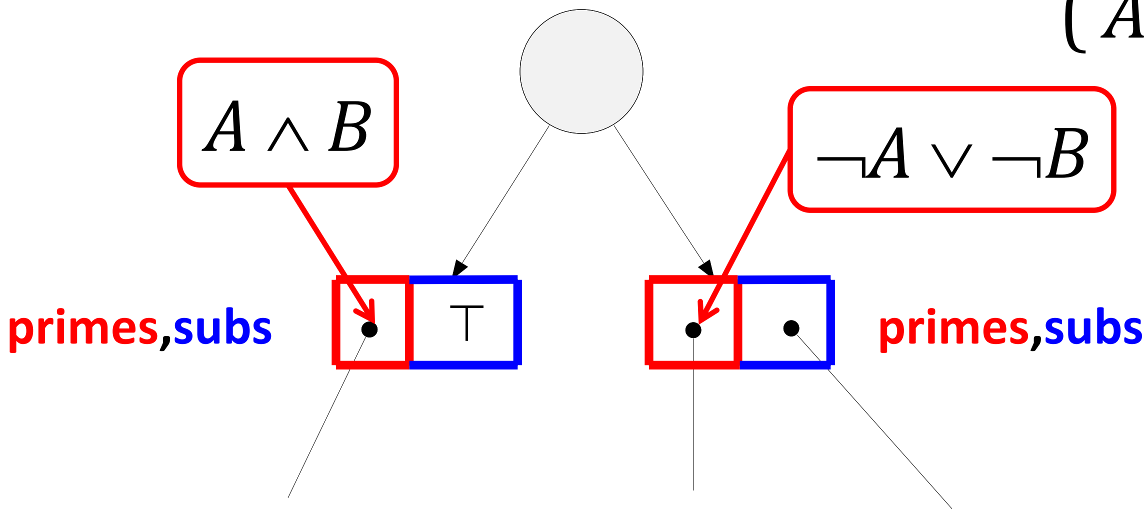
$(A \wedge B) \vee (C \wedge D)$

$\boldsymbol{A = t, B = f,}$
$\boldsymbol{C = t, D = t}$



Basing Decisions on Sentences

$$f(A, B, C, D) = (A \wedge B) \vee (C \wedge D)$$



In an (**X**,**Y**)-partition:

$$f(\mathbf{X}, \mathbf{Y}) = p_1(\mathbf{X})\, s_1(\mathbf{Y}) \vee \ldots \vee p_n(\mathbf{X})\, s_n(\mathbf{Y})$$

primes are ***mutually exclusive***, ***exhaustive*** and not false

Compression and Canonicity

- An (**X**,**Y**)-partition:

  $$f(\mathbf{X}, \mathbf{Y}) = p_1(\mathbf{X})s_1(\mathbf{Y}) \vee \ldots \vee p_n(\mathbf{X})s_n(\mathbf{Y})$$

  is ***compressed*** when the subs are distinct:

  $$s_i(\mathbf{Y}) \neq s_j(\mathbf{Y}) \text{ if } i \neq j$$

- f(**X**,**Y**) has a **unique** compressed **(X,Y)**-partition
- For fixed **X**,**Y** throughout the SDD (i.e. a vtree), compressed SDDs* are **canonical**!

* requires some additional maintenance (pruning/normalization)

Compression

$$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$

$$\mathbf{X} = \{A, B\}, \quad \mathbf{Y} = \{C, D\}$$

Compression

$$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$

$$\mathbf{X} = \{A, B\}, \quad \mathbf{Y} = \{C, D\}$$

prime	sub
$A \wedge B$	
$A \wedge \overline{B}$	
$\overline{A} \wedge B$	
$\overline{A} \wedge \overline{B}$	

Compression

$$f = (A \land B) \lor (B \land C) \lor (C \land D)$$

$$\mathbf{X} = \{A, B\}, \quad \mathbf{Y} = \{C, D\}$$

prime	sub
$A \land B$	true
$A \land \overline{B}$	$C \land D$
$\overline{A} \land B$	C
$\overline{A} \land \overline{B}$	$C \land D$

Compression

$$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$

$$\mathbf{X} = \{A, B\}, \quad \mathbf{Y} = \{C, D\}$$

prime	sub
$A \wedge B$	true
$A \wedge \overline{B}$	$C \wedge D$
$\overline{A} \wedge B$	C
$\overline{A} \wedge \overline{B}$	$C \wedge D$

prime	sub
$A \wedge B$	true
$\overline{A} \wedge B$	C
$\overline{B}$	$C \wedge D$

Compression

$$f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$$

$$\mathbf{X} = \{A, B\}, \quad \mathbf{Y} = \{C, D\}$$

prime	sub
$A \wedge B$	true
$A \wedge \overline{B}$	$C \wedge D$
$\overline{A} \wedge B$	C
$\overline{A} \wedge \overline{B}$	$C \wedge D$

$\vee$

prime	sub
$A \wedge B$	true
$\overline{A} \wedge B$	C
$\overline{B}$	$C \wedge D$

Is Apply for SDDs Polytime?

Algorithm 1 Apply$(\alpha, \beta, \circ)$

if α and β are constants or literals **then**
  return $\alpha \circ \beta$ *// result is a constant or literal*
else if Cache$(\alpha, \beta, \circ) \neq$ nil **then**
  return Cache$(\alpha, \beta, \circ)$ *// has been computed before*
else
  $\gamma \leftarrow \{\}$
  for all elements (p_i, s_i) in α **do**
    for all elements (q_j, r_j) in β **do**
      $p \leftarrow$ Apply$(p_i, q_j, \wedge)$
      if p is consistent **then**
        $s \leftarrow$ Apply$(s_i, r_j, \circ)$
        add element (p, s) to γ

// get unique decision node and return it
  return Cache$(\alpha, \beta, \circ) \leftarrow$ UniqueD(γ)

Is Apply for SDDs Polytime?

Algorithm 1 Apply($\alpha, \beta, \circ$)

if α and β are constants or literals **then**
  return $\alpha \circ \beta$ *// result is a constant or literal*
else if Cache($\alpha, \beta, \circ$) $\neq$ nil **then**
  return Cache($\alpha, \beta, \circ$) *// has been computed before*
else
  $\gamma \leftarrow \{\}$
  for all elements (p_i, s_i) in α **do**
    for all elements (q_j, r_j) in β **do**
      $p \leftarrow$ Apply($p_i, q_j, \wedge$)
      if p is consistent **then**
        $s \leftarrow$ Apply($s_i, r_j, \circ$)
        add element (p, s) to γ

// get unique decision node and return it
  return Cache($\alpha, \beta, \circ$) $\leftarrow$ UniqueD(γ)

- $|\alpha|$x$|\beta|$ recursive calls
- Polytime!

Is Apply for SDDs Polytime?

Algorithm 1 $\texttt{Apply}(\alpha, \beta, \circ)$

if α and β are constants or literals **then**
  return $\alpha \circ \beta$ *// result is a constant or literal*
else if $\texttt{Cache}(\alpha, \beta, \circ) \neq \texttt{nil}$ **then**
  return $\texttt{Cache}(\alpha, \beta, \circ)$ *// has been computed before*
else
  $\gamma \leftarrow \{\}$
  for all elements (p_i, s_i) in α **do**
    for all elements (q_j, r_j) in β **do**
      $p \leftarrow \texttt{Apply}(p_i, q_j, \wedge)$
      if p is consistent **then**
        $s \leftarrow \texttt{Apply}(s_i, r_j, \circ)$
        add element (p, s) to γ

// get unique decision node and return it
  return $\texttt{Cache}(\alpha, \beta, \circ) \leftarrow \texttt{UniqueD}(\gamma)$

- $|\alpha| \text{x} |\beta|$ recursive calls
- Polytime!
- But what about compression/canonicity?

Is Apply for SDDs Polytime?

Algorithm 1 $\texttt{Apply}(\alpha, \beta, \circ)$

if α and β are constants or literals **then**
    return $\alpha \circ \beta$ *// result is a constant or literal*
else if $\texttt{Cache}(\alpha, \beta, \circ) \neq \texttt{nil}$ **then**
    return $\texttt{Cache}(\alpha, \beta, \circ)$ *// has been computed before*
else
    $\gamma \leftarrow \{\}$
    for all elements (p_i, s_i) in α **do**
        for all elements (q_j, r_j) in β **do**
            $p \leftarrow \texttt{Apply}(p_i, q_j, \wedge)$
            if p is consistent **then**
                $s \leftarrow \texttt{Apply}(s_i, r_j, \circ)$
                add element (p, s) to γ
    (optionally) $\gamma \leftarrow \texttt{Compress}(\gamma)$ *// compression*
    // get unique decision node and return it
    return $\texttt{Cache}(\alpha, \beta, \circ) \leftarrow \texttt{UniqueD}(\gamma)$

- Polytime Apply?
- Open question answered in this paper

Theoretical Results

Theorem:

There exists a class of Boolean functions $f_m(X_1,\ldots,X_m)$ such that f_m has an SDD of size $O(m^2)$, yet the canonical SDD of f_m has size $\Omega(2^m)$.

Notation	Transformation	SDD	Canonical SDD
CD	conditioning	$\surd$	$\bullet$
FO	forgetting	$\bullet$	$\bullet$
SFO	singleton forgetting	$\surd$	$\bullet$
$\wedge$**C**	conjunction	$\bullet$	$\bullet$
$\wedge$**BC**	bounded conjunction	$\surd$	$\bullet$
$\vee$**C**	disjunction	$\bullet$	$\bullet$
$\vee$**BC**	bounded disjunction	$\surd$	$\bullet$
$\neg$**C**	negation	$\surd$	$\surd$

Two options

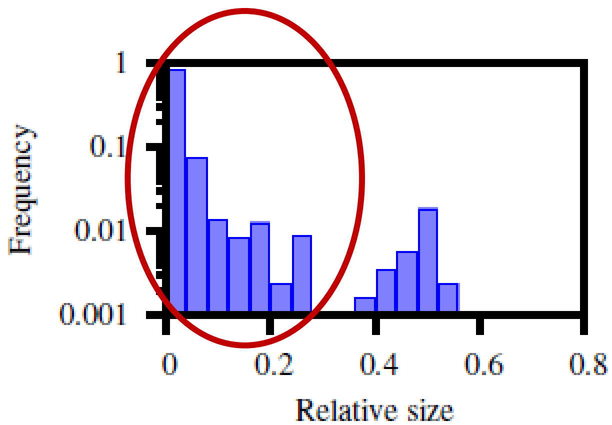
1. Enable compression
   – No polytime Apply
   – Canonicity
2. Disable compression
   – Polytime Apply
   – No Canonicity

What should we do? Popular belief:
Choose polytime Apply, or circuits blow up!

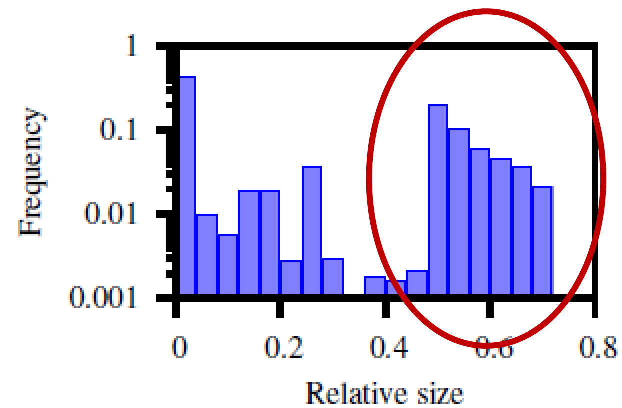
Empirical Results

Name	Variables	Clauses	SDD Size			Compilation Time		
			Compressed SDDs+s	Compressed SDDs	Uncompressed SDDs	Compressed SDDs+s	Compressed SDDs	Uncompressed SDDs
C17	17	30	99	171	286	0.00	0.00	0.00
majority	14	35	123	193	384	0.00	0.00	0.00
b1	21	50	166	250	514	0.00	0.00	0.00
cm152a	20	49	149	3,139	18,400	0.01	0.01	0.02
cm82a	25	62	225	363	683	0.01	0.00	0.00
cm151a	44	100	614	1,319	24,360	0.04	0.00	0.04
cm42a	48	110	394	823	276,437	0.03	0.00	0.10
cm138a	50	114	463	890	9,201,336	0.02	0.01	109.05
decod	41	122	471	810	1,212,302	0.04	0.01	1.40
tcon	65	136	596	1,327	618,947	0.05	0.00	0.33
parity	61	135	549	978	2,793	0.02	0.00	0.00
cmb	62	147	980	2,311	81,980	0.12	0.02	0.06
cm163a	68	157	886	1,793	21,202	0.06	0.00	0.02
pcle	66	156	785	1,366	n/a	0.07	0.01	n/a
x2	62	166	785	1,757	12,150,626	0.08	0.02	19.87
cm85a	77	176	1,015	2,098	19,657	0.08	0.01	0.03
cm162a	73	173	907	2,050	153,228	0.08	0.01	0.16
cm150a	84	202	1,603	5,805	17,265,164	0.16	0.06	60.37
pcler8	98	220	1,518	4,335	15,532,667	0.18	0.05	33.32
cu	94	235	1,466	5,789	n/a	0.19	0.10	n/a
pm1	105	245	1,810	3,699	n/a	0.27	0.05	n/a
mux	73	240	1,825	6,517	n/a	0.19	0.09	n/a
cc	115	265	1,451	6,938	n/a	0.22	0.04	n/a
unreg	149	336	3,056	668,531	n/a	0.66	263.06	n/a
ldd	145	414	1,610	2,349	n/a	0.23	0.10	n/a
count	185	425	4,168	51,639	n/a	1.05	0.24	n/a
comp	197	475	2,212	4,500	205,105	0.24	0.01	0.22
f51m	108	511	3,290	6,049	n/a	0.52	0.32	n/a
my_adder	212	612	2,793	4,408	35,754	0.24	0.02	0.04
cht	205	650	4,832	13,311	n/a	1.24	0.36	n/a

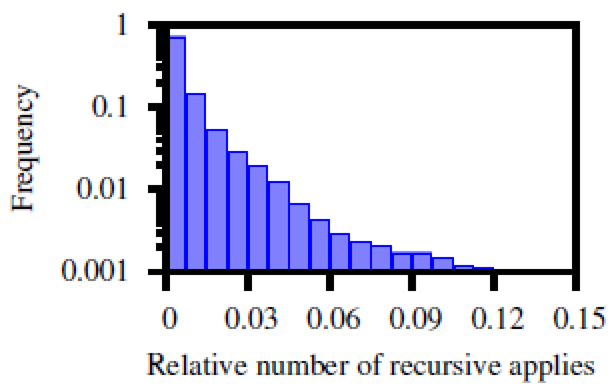
Empirical Results



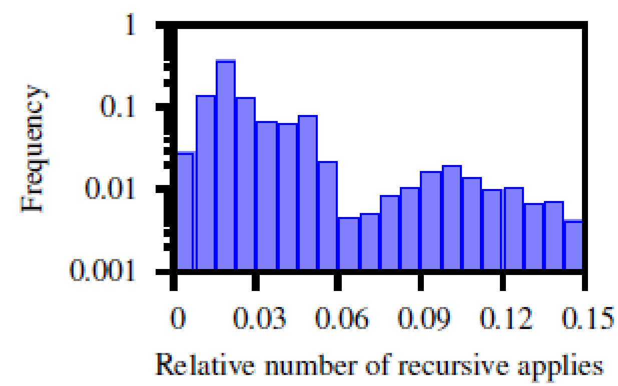
(a) Compressed SDDs



(b) Uncompressed SDDs



(a) Compressed SDDs



(b) Uncompressed SDDs

What We Know Now

- Canonical SDDs have **no polytime Apply**!
- Yet they work!
  Outperform OBDDs and non-canonical SDDs
- We argue: **Canonicity** is more important
  - Facilitates caching and minimization (vtree search)
- Questions common wisdom

Thanks