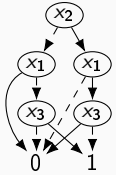
Separating Incremental and Non-Incremental Bottom-Up Compilation

Alexis de Colnet

Knowledge Compilation

Knowledge compilation (KC) deals with representations of (Boolean) functions.

$$
\begin{aligned}
& (x_1 \vee \neg x_2) \\
\wedge\ & (\neg x_1 \vee x_2) \\
\wedge\ & (x_1 \vee x_2 \vee x_3) \\
\wedge\ & (\neg x_1 \vee \neg x_2 \vee \neg x_3)
\end{aligned}
$$



$\mathcal{L}_0$ and $\mathcal{L}_1$: two classes of function representations, also called languages.

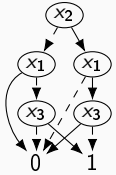
Given a function F represented in $\mathcal{L}_0$, compute a representation Σ in $\mathcal{L}_1$ that is equivalent to F.

$$F \equiv \Sigma \qquad \Leftrightarrow \qquad var(F) = var(\Sigma) = X \text{ and } \forall a \in \{0,1\}^X, F(a) = \Sigma(a)$$

Knowledge Compilation

Knowledge compilation (KC) deals with representations of (Boolean) functions.

$$
\begin{aligned}
&(x_1 \vee \neg x_2) \\
\wedge\ &(\neg x_1 \vee x_2) \\
\wedge\ &(x_1 \vee x_2 \vee x_3) \\
\wedge\ &(\neg x_1 \vee \neg x_2 \vee \neg x_3)
\end{aligned}
\quad \longrightarrow \text{compilation} \longrightarrow
$$



$\mathcal{L}_0$ and $\mathcal{L}_1$: two classes of function representations, also called languages.

Given a function F represented in $\mathcal{L}_0$, compute a representation Σ in $\mathcal{L}_1$ that is equivalent to F.

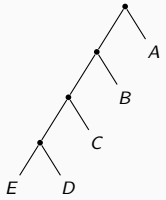
$$F \equiv \Sigma \quad \Leftrightarrow \quad var(F) = var(\Sigma) = X \text{ and } \forall a \in \{0,1\}^X, F(a) = \Sigma(a)$$

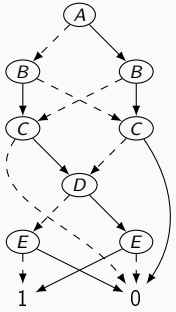
OBDD [Bryant, IEEE Trans. Comp. 1986]

Ordered Binary Decision Diagrams

Support many queries in linear time (model counting, clausal entailment, model enumeration...)

Structured by a linear vtree





Conjunction of 2 OBDDs with the same structure in quadratic time

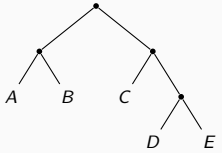
Conjunction of 2 OBDDs with the different structure is generally intractable

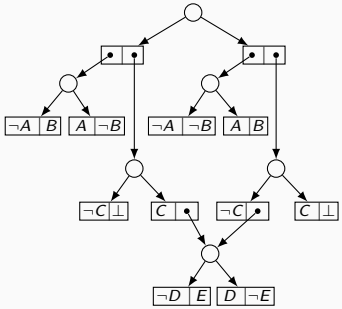
SDD [Darwiche, IJCAI 2011]

Sentencial Decision Diagrams

Support many queries in linear time (model counting, clausal entailment, model enumeration...)

Structured by a ~~linear~~ vtree



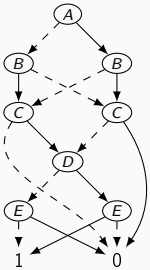


Conjunction of 2 SDDs with the same structure in quadratic time

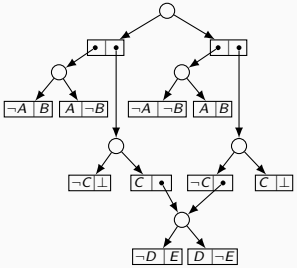
Conjunction of 2 SDDs with the different structure is generally intractable

Some Compilation Languages

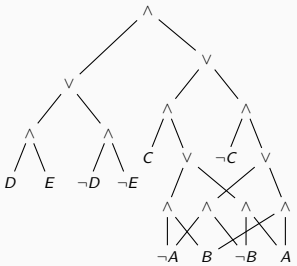
OBDD



SDD



strDNNF



strDNNFs [Pipatsrisawat and Darwiche, AAAI 2008] are strictly more succinct than SDDs, which are strictly more succinct than OBDDs [Bova, AAAI 2016]

All structured by vtrees + tractable conjunction of any two diagrams/circuits with the same structure.

This makes bottom-up compilation to these languages possible.

An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

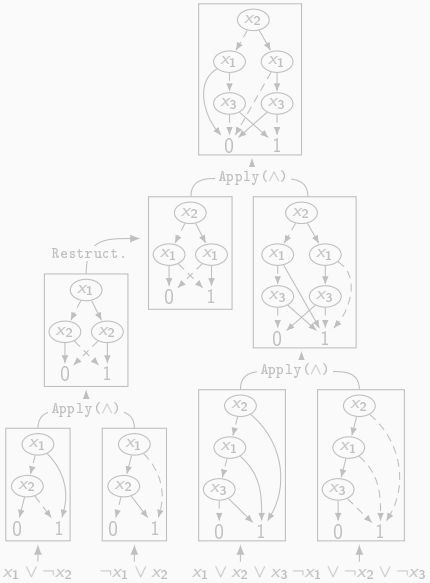
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

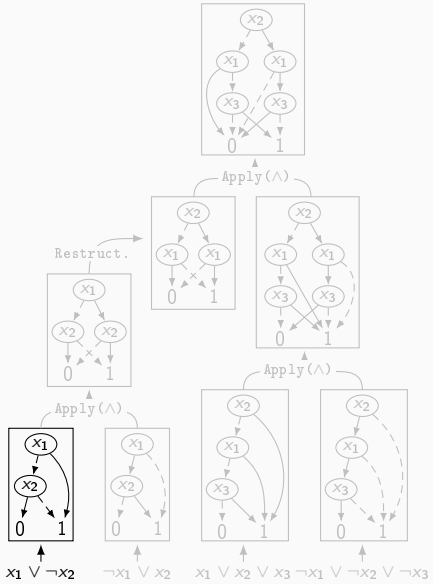
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

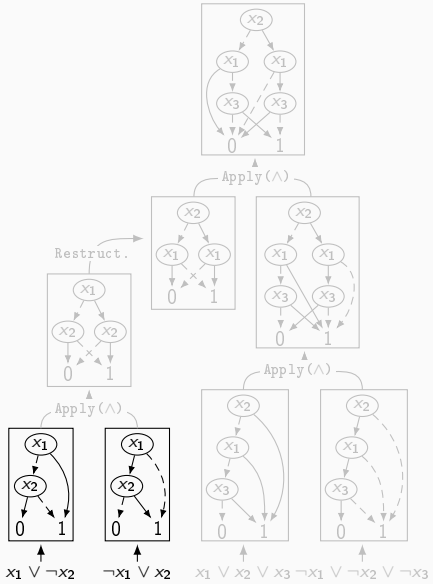
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

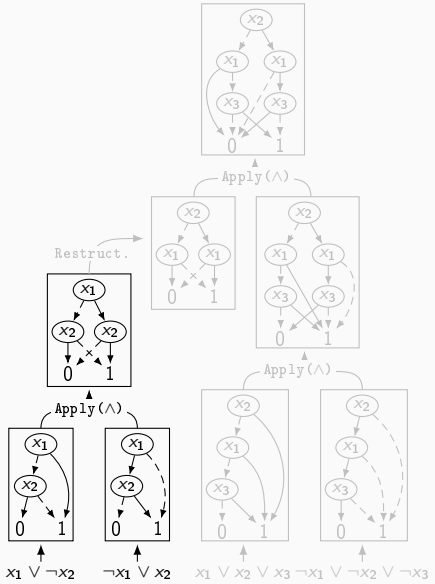
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

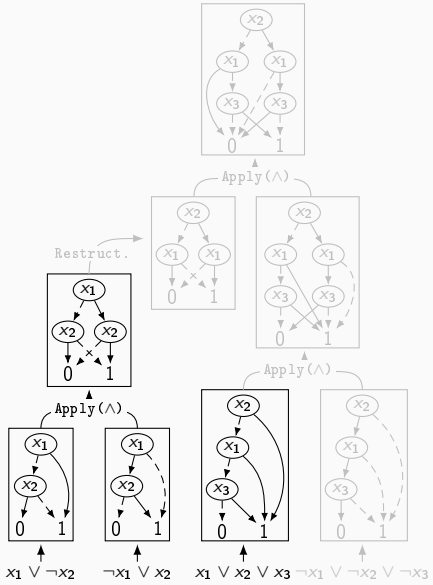
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

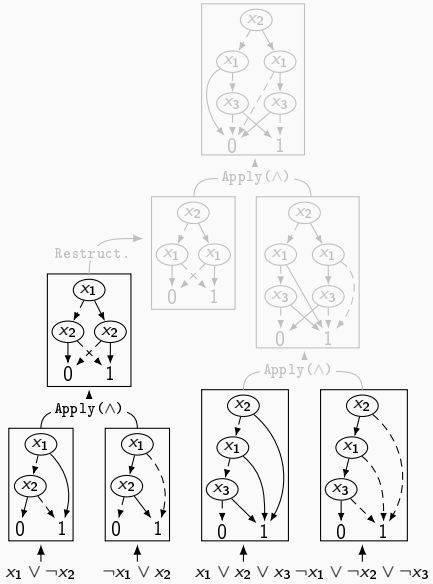
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

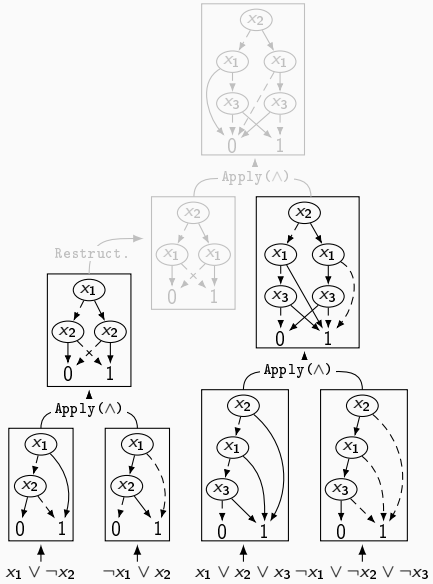
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

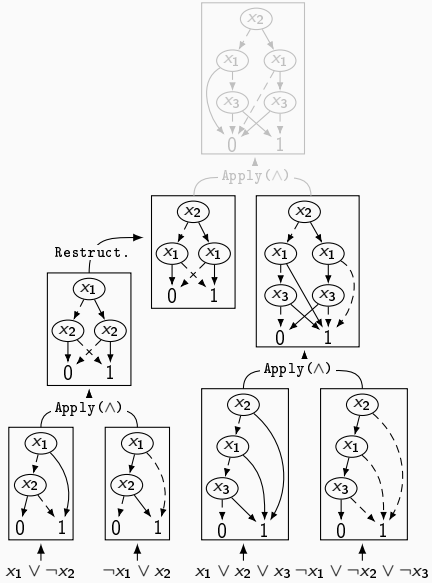
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



An Example of BU Compilation

BU compilation to OBDD of the formula

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_1 = \texttt{Compile}(x_1 \vee \neg x_2)$

$\Sigma_2 = \texttt{Compile}(\neg x_1 \vee x_2)$

$\Sigma_3 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$

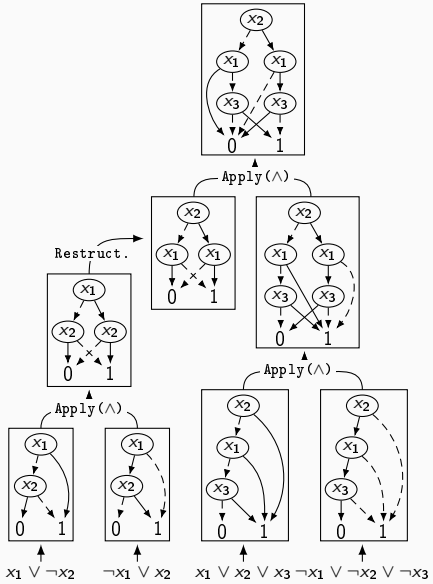
$\Sigma_4 = \texttt{Compile}(x_1 \vee x_2 \vee x_3)$

$\Sigma_5 = \texttt{Compile}(\neg x_1 \vee \neg x_2 \vee \neg x_3)$

$\Sigma_6 = \texttt{Apply}(\Sigma_4, \Sigma_5, \wedge)$

$\Sigma_7 = \texttt{Restructure}(\Sigma_3)$

$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$



The Theoretical Framework for BU Compilation

For me, a bottom-up compilation of a CNF formula/system of constraints F to a language $\mathcal{L}$ (OBDD or SDD or strDNNF) looks like this:

$$(\Sigma_1, I_1), (\Sigma_2, I_2), \ldots, (\Sigma_N, I_N)$$

where $F \equiv \Sigma_N$ and, for every $i \in [N]$, $\Sigma_i \in \mathcal{L}$, and either

- $I_i : \Sigma_i = \texttt{Compile}(C)$ for some $C \in F$, then $\Sigma_i \equiv C$
- $I_i : \Sigma_i = \texttt{Restructure}(\Sigma_j)$ for some $j < i$, then $\Sigma_i \equiv \Sigma_j$
- $I_i : \Sigma_i = \texttt{Apply}(\Sigma_j, \Sigma_k, \wedge)$ for some $j, k < i$, then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$ and Σ_i, Σ_j, Σ_k have the same structure/vtree/variable ordering...

In the worst cases, some Σ_i can get very large.

The Theoretical Framework for BU Compilation

For me, a bottom-up compilation of a CNF formula/system of constraints F to a language $\mathcal{L}$ (OBDD or SDD or strDNNF) looks like this:

$$(\Sigma_1, I_1), (\Sigma_2, I_2), \ldots, (\Sigma_N, I_N)$$

where $F \equiv \Sigma_N$ and, for every $i \in [N]$, $\Sigma_i \in \mathcal{L}$, and either

- $I_i : \Sigma_i = \texttt{Compile}(C)$ for some $C \in F$, then $\Sigma_i \equiv C$
- $I_i : \Sigma_i = \texttt{Restructure}(\Sigma_j)$ for some $j < i$, then $\Sigma_i \equiv \Sigma_j$
- $I_i : \Sigma_i = \texttt{Apply}(\Sigma_j, \Sigma_k, \wedge)$ for some $j, k < i$, then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$ and Σ_i, Σ_j, Σ_k have the same structure/vtree/variable ordering...

In the worst cases, some Σ_i can get very large.

The Theoretical Framework for BU Compilation

For me, a bottom-up compilation of a CNF formula/system of constraints F to a language $\mathcal{L}$ (OBDD or SDD or strDNNF) looks like this:

$$(\Sigma_1, I_1), (\Sigma_2, I_2), \ldots, (\Sigma_N, I_N)$$

where $F \equiv \Sigma_N$ and, for every $i \in [N]$, $\Sigma_i \in \mathcal{L}$, and either

- $I_i : \Sigma_i = \texttt{Compile}(C)$ for some $C \in F$, then $\Sigma_i \equiv C$
- $I_i : \Sigma_i = \texttt{Restructure}(\Sigma_j)$ for some $j < i$, then $\Sigma_i \equiv \Sigma_j$
- $I_i : \Sigma_i = \texttt{Apply}(\Sigma_j, \Sigma_k, \wedge)$ for some $j, k < i$, then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$ and Σ_i, Σ_j, Σ_k have the same structure/vtree/variable ordering...

In the worst cases, some Σ_i can get very large.

The Theoretical Framework for BU Compilation

For me, a bottom-up compilation of a CNF formula/system of constraints F to a language $\mathcal{L}$ (OBDD or SDD or strDNNF) looks like this:

$$(\Sigma_1, I_1), (\Sigma_2, I_2), \ldots, (\Sigma_N, I_N)$$

where $F \equiv \Sigma_N$ and, for every $i \in [N]$, $\Sigma_i \in \mathcal{L}$, and either

- $I_i : \Sigma_i = \texttt{Compile}(C)$ for some $C \in F$, then $\Sigma_i \equiv C$
- $I_i : \Sigma_i = \texttt{Restructure}(\Sigma_j)$ for some $j < i$, then $\Sigma_i \equiv \Sigma_j$
- $I_i : \Sigma_i = \texttt{Apply}(\Sigma_j, \Sigma_k, \wedge)$ for some $j, k < i$, then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$ and Σ_i, Σ_j, Σ_k have the same structure/vtree/variable ordering...

In the worst cases, some Σ_i can get very large.

Different Flavors of BU Compilations

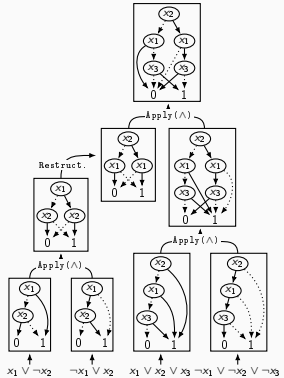
The framework gives complete freedom on the order in which the `Applys` are performed.

Say one wants to compile $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ to $\mathcal{L}$.

$$\Sigma_1 = \texttt{Compile}(C_1)$$
$$\Sigma_2 = \texttt{Compile}(C_2)$$
$$\Sigma_3 = \texttt{Compile}(C_3)$$
$$\Sigma_4 = \texttt{Compile}(C_4)$$

The compilation might pursue like this:

$$\Sigma_5 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$$
$$\Sigma_6 = \texttt{Apply}(\Sigma_3, \Sigma_4, \wedge)$$
$$\Sigma_7 = \texttt{Apply}(\texttt{Restructure}(\Sigma_5), \Sigma_6, \wedge)$$



Different Flavors of BU Compilations

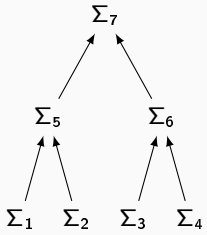
The framework gives complete freedom on the order in which the `Applys` are performed.

Say one wants to compile $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ to $\mathcal{L}$.

$\Sigma_1 = \texttt{Compile}(C_1)$
$\Sigma_2 = \texttt{Compile}(C_2)$
$\Sigma_3 = \texttt{Compile}(C_3)$
$\Sigma_4 = \texttt{Compile}(C_4)$

The compilation might pursue like this:

$\Sigma_5 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$
$\Sigma_6 = \texttt{Apply}(\Sigma_3, \Sigma_4, \wedge)$
$\Sigma_7 = \texttt{Apply}(\Sigma_5, \Sigma_6, \wedge)$



Different Flavors of BU Compilations

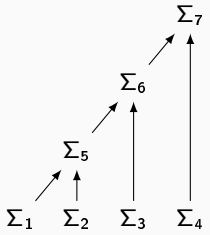
The framework gives complete freedom on the order in which the `Applys` are performed.

Say one wants to compile $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ to $\mathcal{L}$.

$$\Sigma_1 = \texttt{Compile}(C_1)$$
$$\Sigma_2 = \texttt{Compile}(C_2)$$
$$\Sigma_3 = \texttt{Compile}(C_3)$$
$$\Sigma_4 = \texttt{Compile}(C_4)$$

Or like that:

$$\Sigma_5 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$$
$$\Sigma_6 = \texttt{Apply}(\Sigma_5, \Sigma_3, \wedge)$$
$$\Sigma_7 = \texttt{Apply}(\Sigma_6, \Sigma_4, \wedge)$$



Different Flavors of BU Compilations

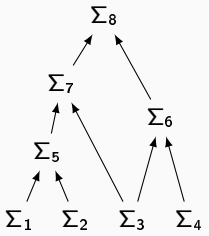
The framework gives complete freedom on the order in which the `Applys` are performed.

Say one wants to compile $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ to $\mathcal{L}$.

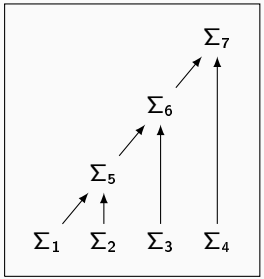
$$\Sigma_1 = \texttt{Compile}(C_1)$$
$$\Sigma_2 = \texttt{Compile}(C_2)$$
$$\Sigma_3 = \texttt{Compile}(C_3)$$
$$\Sigma_4 = \texttt{Compile}(C_4)$$

Or even like that:

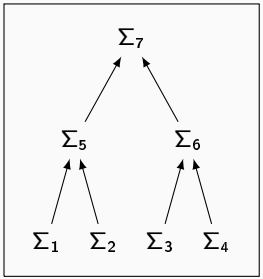
$$\Sigma_5 = \texttt{Apply}(\Sigma_1, \Sigma_2, \wedge)$$
$$\Sigma_6 = \texttt{Apply}(\Sigma_3, \Sigma_4, \wedge)$$
$$\Sigma_7 = \texttt{Apply}(\Sigma_5, \Sigma_3, \wedge)$$
$$\Sigma_8 = \texttt{Apply}(\Sigma_6, \Sigma_7, \wedge)$$



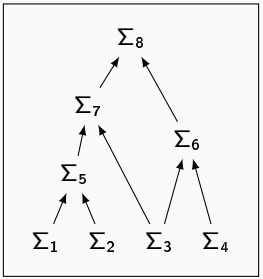
Different Flavors of BU Compilations



Incremental
BU compilation



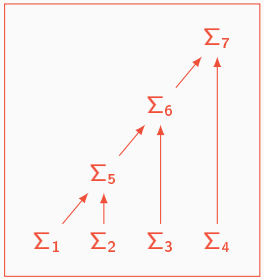
Tree-like
BU compilation



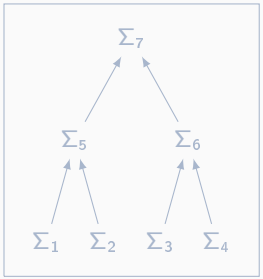
General (DAG-like)
BU compilation

In incremental BU all `Applys` are of the form $\Sigma_i = \texttt{Apply}(\Sigma_j, \texttt{Compile}(C), \wedge)$ (or just $\Sigma_i = \texttt{Apply}(\Sigma_j, C, \wedge)$).

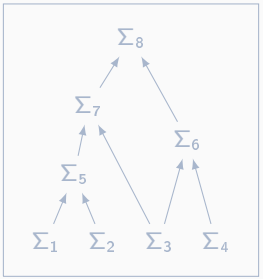
Different Flavors of BU Compilations



Incremental
BU compilation



Tree-like
BU compilation



General (DAG-like)
BU compilation

In incremental BU all `Applys` are of the form $\Sigma_i = \texttt{Apply}(\Sigma_j, \texttt{Compile}(C), \wedge)$ (or just $\Sigma_i = \texttt{Apply}(\Sigma_j, C, \wedge)$).

Incremental BU Compilation in Practice

Incremental BU compilation has been used for OBDDs [Narodytska and Walsh, IJCAI 2007] and SDDs [Choi and Darwiche, AAAI 2013] and for pseudo-Boolean functions [SALADD compilers][1]

Incremental BU compilation for CNF is attractive in practice.

- the problem of ordering the `Applys` is reduced to that of ordering the input clauses
- the $\texttt{Apply}(\Sigma, C, \wedge)$ are feasible in linear time
- the `Restructure` operation has a unique purpose: reducing the size of the core circuit

Some BU compilation strategies are "close to incremental". If a BU compilation is such that, every `Apply` instruction is either

- an $\texttt{Apply}(\Sigma, C, \wedge)$ for some clause C, or
- an $\texttt{Apply}(\Sigma, \Sigma', \wedge)$ with $var(\Sigma) \cap var(\Sigma') = \emptyset$

then the BU compilation can be turned into an incremental BU compilation in polynomial time.

[1] https://www.irit.fr/ Helene.Fargier/BR4CP/CompilateurSALADD.html

Contributions

The paper shows an exponential separation between incremental BU compilation and general BU compilation.

There exists a class $\mathcal{F}$ of CNF formulas that admit polynomial-size BU compilation into OBDD, but whose incremental BU compilations into OBDD/SDD/str-DNNF all generate intermediate circuits of size exponential in the number of variables.

So there are infinitely many CNF formulas that are

1. "easy" to compile in general BU fashion
2. hard to compile in incremental BU fashion

A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

For F_1 and F_2 any two CNF formulas, $F_1 \veebar F_2$ is the CNF formula $\bigwedge_{C_1 \in F_1} \bigwedge_{C_2 \in F_2} (C_1 \vee C_2)$.

$$F_1 \veebar F_2 \equiv F_1 \vee F_2 \qquad \text{and} \qquad |F_1 \veebar F_2| = |F_1|.|F_2|$$

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile (even incrementally)

then $F_1 \veebar F_2$ is easy to BU compile (non-incrementally).

A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile

then $F_1 \veebar F_2$ is easy to BU compile.

$$\bigwedge_{C_2 \in F_2} \bigwedge_{C_1 \in F_1} (C_1 \vee C_2)$$

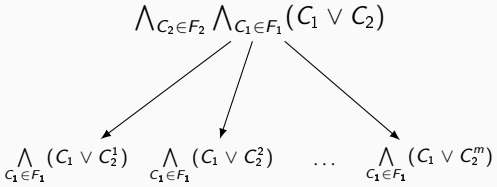
A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile

then $F_1 \veebar F_2$ is easy to BU compile.



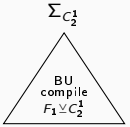
A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile

then $F_1 \veebar F_2$ is easy to BU compile.



$$\bigwedge_{C_1 \in F_1} (C_1 \vee C_2^1) \quad \bigwedge_{C_1 \in F_1} (C_1 \vee C_2^2) \quad \ldots \quad \bigwedge_{C_1 \in F_1} (C_1 \vee C_2^m)$$

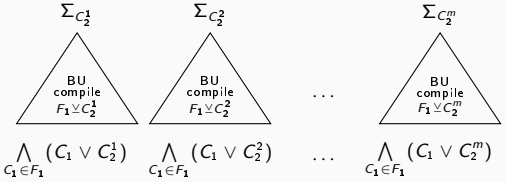
A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile

then $F_1 \veebar F_2$ is easy to BU compile.



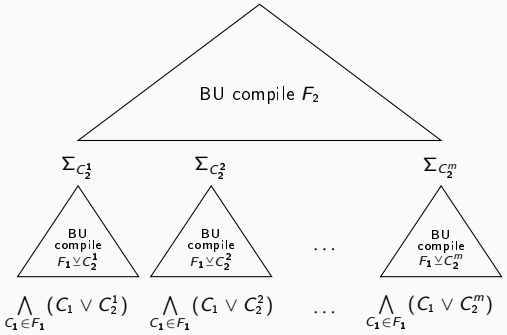
A Quick Peek into the Proof

1 CNF formulas that are "easy" to compile with general BU compilation

Suppose

- F_1 is unsatisfiable and,
- F_1 and F_2 are easy to BU compile

then $F_1 \veebar F_2$ is easy to BU compile.



2 CNF formulas that are hard to compile incrementally

For the incremental BU compilation of $F_1 \veebar F_2$, consider the last `Apply`:

$$(\Sigma_1, I_1),\ (\Sigma_2, I_2),\ \ldots,\ \underbrace{(\Sigma_N, I_N)}_{I_N : \Sigma_N = \texttt{Apply}(\Sigma_{N-1}, C_1 \vee C_2, \wedge)}$$

Suppose both F_1 and F_2 are unsatisfiable, then

$$\Sigma_{N-1} \equiv (F_1 \setminus C_1) \wedge (F_2 \setminus C_2).$$

So compiling $F_1 \veebar F_2$ incrementally is at least as hard as compiling $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$.

A (not so) Quick Peek into the Proof

For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

A (not so) Quick Peek into the Proof

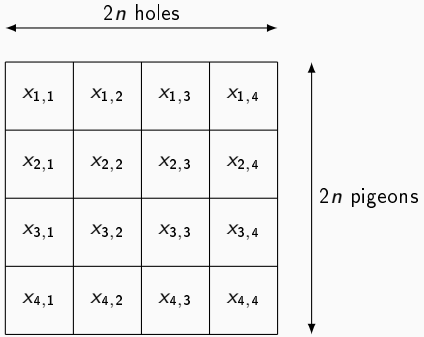
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

$\mathsf{P}(X) =$ exactly one pigeon per hole

$\mathsf{H}(X) =$ exactly one hole for each pigeon

$\mathsf{ODD}(X) =$ there is an odd number of pigeon in total



A (not so) Quick Peek into the Proof

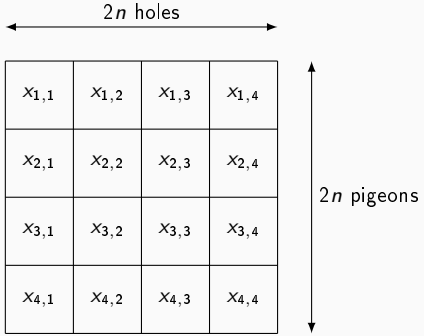
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$

$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$

$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$



A (not so) Quick Peek into the Proof

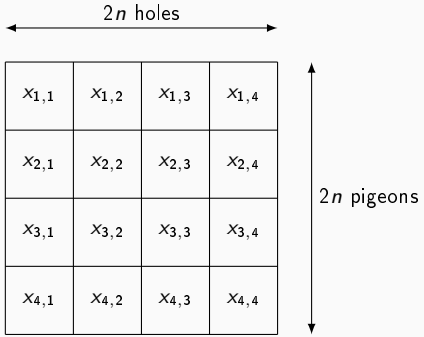
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$
$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$
$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$

$F_1(X, Y) = F_{\mathsf{P}}(X) \wedge F_{\mathsf{ODD}}(X, Y)$
$F_2(X, Z) = F_{\mathsf{H}}(X) \wedge F_{\mathsf{ODD}}(X, Z)$

(⚠ not exactly the functions used in the paper!)



A (not so) Quick Peek into the Proof

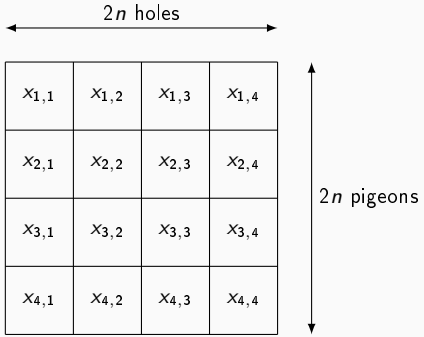
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable ✓
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$
$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$
$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$

$F_1(X, Y) = F_{\mathsf{P}}(X) \wedge F_{\mathsf{ODD}}(X, Y)$
$F_2(X, Z) = F_{\mathsf{H}}(X) \wedge F_{\mathsf{ODD}}(X, Z)$

(⚠ not exactly the functions used in the paper!)



A (not so) Quick Peek into the Proof

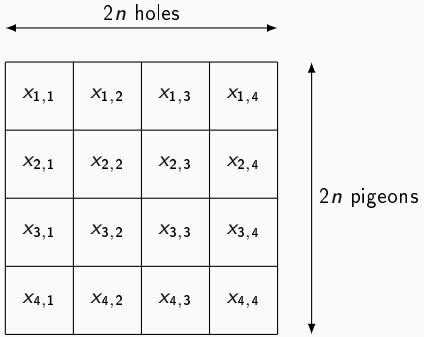
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable ✓
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$ ✓
- $(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$
$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$
$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$

$F_1(X, Y) = F_{\mathsf{P}}(X) \wedge F_{\mathsf{ODD}}(X, Y)$
$F_2(X, Z) = F_{\mathsf{H}}(X) \wedge F_{\mathsf{ODD}}(X, Z)$

(⚠ not exactly the functions used in the paper!)



A (not so) Quick Peek into the Proof

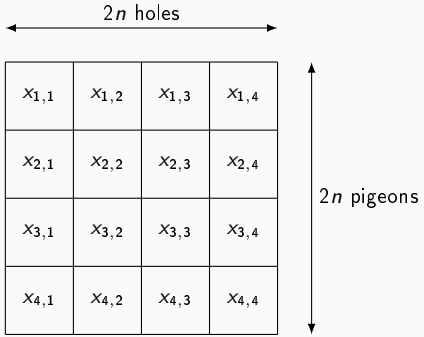
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable ✓
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$ ✓
- ~~$(F_1 \setminus C_1) \wedge (F_2 \setminus C_2)$ has exponential size in $\mathcal{L}$ for every $C_1 \in F_1$, $C_2 \in F_2$~~

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$
$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$
$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$

$F_1(X, Y) = F_{\mathsf{P}}(X) \wedge F_{\mathsf{ODD}}(X, Y)$
$F_2(X, Z) = F_{\mathsf{H}}(X) \wedge F_{\mathsf{ODD}}(X, Z)$

(⚠ not exactly the functions used in the paper!)



A (not so) Quick Peek into the Proof

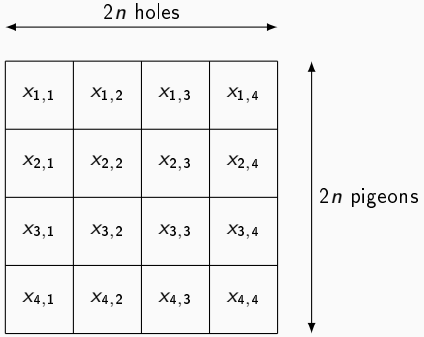
For $\mathcal{L} \in \{\text{OBDD, SDD, strDNNF}\}$, we want F_1 and F_2 such that

- F_1 and F_2 are unsatisfiable ✓
- F_1 and F_2 are "easy" to BU compile in $\mathcal{L}$ ✓
- $\text{PHP}_{2n-\Delta}^{2n-\Delta}$ is hidden in every incremental BU compilation of $F_1 \veebar F_2$ ✓

$F_{\mathsf{P}}(X) =$ CNF representation of $\mathsf{P}(X)$
$F_{\mathsf{H}}(X) =$ CNF representation of $\mathsf{H}(X)$
$F_{\mathsf{ODD}}(X, Y) =$ standard CNF encoding of $\mathsf{ODD}(X)$

$F_1(X, Y) = F_{\mathsf{P}}(X) \wedge F_{\mathsf{ODD}}(X, Y)$
$F_2(X, Z) = F_{\mathsf{H}}(X) \wedge F_{\mathsf{ODD}}(X, Z)$

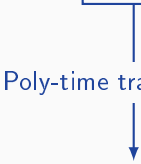
(⚠ not exactly the functions used in the paper!)



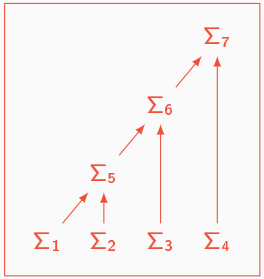
A ~~Quick~~ Long Peek into the Proof

In every incremental BU compilation of $F_1(X, Y) \veebar F_2(X, Z)$, there exists k, a small Δ and an $2n - \Delta \times 2n - \Delta$ submatrix X' of X such that

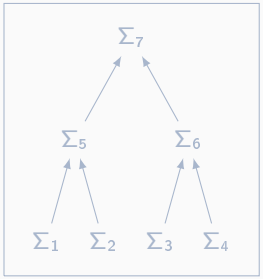
$$(\Sigma_1, I_1), (\Sigma_2, I_2), \ldots, \boxed{(\Sigma_k, I_k),} \ldots, (\Sigma_N, I_N)$$



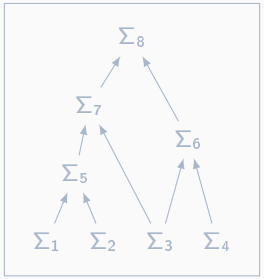
Every OBDD/SDD/DNNF representing $PHP^n_n(X)$ has size $2^{\Omega(n)} poly(1/n)$.
[Wegener, 2000; Wallon and Mengel, JAIR 2020; de C, SAT 2020]



Incremental
BU compilation



Tree-like
BU compilation



General (DAG-like)
BU compilation

- BU compilers use incremental (clause-by-clause) approaches
- Incremental compilations are exponentially less compact than general BU compilations
- Can we create compilers that compile in tree-like BU fashion?
- Can we separate tree-like BU compilation from DAG-lake BU compilation?