Connecting Width and Structure in Knowledge Compilation

Antoine Amarilli[1], **Mikaël Monet**[1,3], Pierre Senellart[1,2,3]

October 24th, 2018 (results from ICDT 2018 paper)

[1]LTCI, Télécom ParisTech, Université Paris-Saclay; Paris, France

[2]École normale supérieure, PSL University; Paris, France

[3]Inria; Paris, France

What is Knowledge Compilation?

- You have a task
  - → Boolean SAT (is there a satisfying assignment?)

What is Knowledge Compilation?

- You have a task
  - → Boolean SAT (is there a satisfying assignment?)
  - → #SAT (model counting) (how many satisfying assignments?)

What is Knowledge Compilation?

- You have a task
  - → Boolean SAT (is there a satisfying assignment?)
  - → #SAT (model counting) (how many satisfying assignments?)
  - → probabilistic evaluation

What is Knowledge Compilation?

- You have a **task**
  - → Boolean SAT (is there a satisfying assignment?)
  - → #SAT (model counting) (how many satisfying assignments?)
  - → probabilistic evaluation
  - → enumeration

What is Knowledge Compilation?

- You have a task
  - → Boolean SAT (is there a satisfying assignment?)
  - → #SAT (model counting) (how many satisfying assignments?)
  - → probabilistic evaluation
  - → enumeration
- Idea: compile the input into a format that is *designed* to solve efficiently your task

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$

$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. } 2} \text{Result}$$

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. 1}} \text{Result}$$

$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. 2}} \text{Result}$$

$$\text{Input class } \mathcal{C}_3 \xrightarrow{\text{Algo. 3}} \text{Result}$$

...

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$

$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. } 2} \text{Result}$$

$$\text{Input class } \mathcal{C}_3 \xrightarrow{\text{Algo. } 3} \text{Result}$$

...

- With knowledge compilation:

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$

$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. } 2} \text{Result}$$

$$\text{Input class } \mathcal{C}_3 \xrightarrow{\text{Algo. } 3} \text{Result}$$

...

- With knowledge compilation:

Input class $\mathcal{C}_1$

Input class $\mathcal{C}_2$ Compilation target for your task $\xrightarrow{\text{Generic algo.}}$ Result

Input class $\mathcal{C}_3$

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$
$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. } 2} \text{Result}$$
$$\text{Input class } \mathcal{C}_3 \xrightarrow{\text{Algo. } 3} \text{Result}$$

...

- With knowledge compilation:

Input class $\mathcal{C}_1$ $\xrightarrow{\text{Algo. } 1'}$ Compilation target for your task $\xrightarrow{\text{Generic algo.}}$ Result

Input class $\mathcal{C}_2$

Input class $\mathcal{C}_3$

Why would I do that?

- Without knowledge compilation

Input class $\mathcal{C}_1 \xrightarrow{\text{Algo. } 1}$ Result

Input class $\mathcal{C}_2 \xrightarrow{\text{Algo. } 2}$ Result

Input class $\mathcal{C}_3 \xrightarrow{\text{Algo. } 3}$ Result

...

- With knowledge compilation:

Input class $\mathcal{C}_1 \xrightarrow{\text{Algo. } 1'}$ Compilation target for your task

Input class $\mathcal{C}_2 \xrightarrow{\text{Algo. } 2'}$ Compilation target for your task $\xrightarrow{\text{Generic algo.}}$ Result

Input class $\mathcal{C}_3$

Why would I do that?

- Without knowledge compilation

$$\text{Input class } \mathcal{C}_1 \xrightarrow{\text{Algo. } 1} \text{Result}$$

$$\text{Input class } \mathcal{C}_2 \xrightarrow{\text{Algo. } 2} \text{Result}$$

$$\text{Input class } \mathcal{C}_3 \xrightarrow{\text{Algo. } 3} \text{Result}$$

...

- With knowledge compilation:

Input class $\mathcal{C}_1$ —Algo. $1'$→ Compilation target for your task

Input class $\mathcal{C}_2$ —Algo. $2'$→ Compilation target for your task

Input class $\mathcal{C}_3$ —Algo. $3'$→ Compilation target for your task

$$\text{Compilation target for your task} \xrightarrow{\text{Generic algo.}} \text{Result}$$

Why would I do that?

- Without knowledge compilation

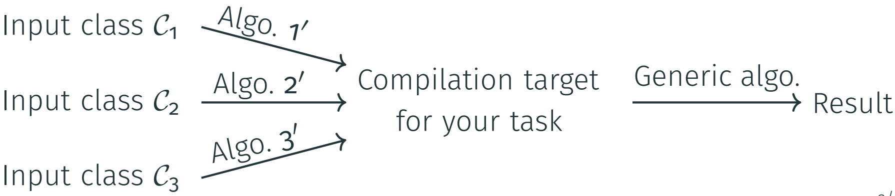
Input class $\mathcal{C}_1 \xrightarrow{\text{Algo. } 1}$ Result

Input class $\mathcal{C}_2 \xrightarrow{\text{Algo. } 2}$ Result

Input class $\mathcal{C}_3 \xrightarrow{\text{Algo. } 3}$ Result

...

- With knowledge compilation: modularity!



Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table Evaluation

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table O(1) Evaluation

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table O(1) Evaluation

SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table	O(1)	Evaluation
	O(n)	SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table	O(1)	Evaluation
	O(n)	SAT
		#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Truth table	O(1)	Evaluation
	O(n)	SAT
	O(n)	#SAT

Studying the compilation targets

- Tradeoffs between:
  - $\rightarrow$ Complexity of compilation (*conciseness* of the compilation target)
  - $\rightarrow$ Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

DNF

Evaluation

SAT

#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

DNF	O(n)	Evaluation
		SAT
		#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

DNF	O(n)	Evaluation
	O(n)	SAT
		#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

DNF

O(n)	Evaluation
O(n)	SAT
#P-hard	#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit Evaluation

SAT

#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit O(n) Evaluation

SAT

#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit	O(n)	Evaluation
	NP-hard	SAT
		#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit	O(n)	Evaluation
	NP-hard	SAT
	#P-hard	#SAT

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit	O(n)	Evaluation
	NP-hard	SAT
	#P-hard	#SAT

→ When can we convert from one target to another?

Studying the compilation targets

- Tradeoffs between:
  - → Complexity of compilation (*conciseness* of the compilation target)
  - → Complexity of solving the task

$$\text{Input} \xrightarrow{\text{Compilation}} \mathcal{C}_{\text{target}} \xrightarrow{\text{Complexity}} \text{Result}$$

Boolean circuit	O(n)	Evaluation
	NP-hard	SAT
	#P-hard	#SAT

→ When can we convert from one target to another?

- We are interested in **#SAT** and **probability evaluation**

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  $\rightarrow$ message passing algorithm for #SAT and probabilistic evaluation

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  - $\rightarrow$ message passing algorithm for #SAT and probabilistic evaluation
    - Links with **Bayesian networks**

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  - → message passing algorithm for #SAT and probabilistic evaluation
    - Links with **Bayesian networks**

Semantics-based:

- **Ordered Binary Decision Diagrams** (OBDDs)/ **Deterministic Structured Decomposable Negation Normal Forms** (d-SDNNFs)

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  → message passing algorithm for #SAT and probabilistic evaluation
    - Links with **Bayesian networks**

Semantics-based:

- **Ordered Binary Decision Diagrams** (OBDDs)/ **Deterministic Structured Decomposable Negation Normal Forms** (d-SDNNFs)
  → #SAT and probabilistic evaluation are easy because these classes have strong semantic constraints

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  - → message passing algorithm for #SAT and probabilistic evaluation
    - Links with **Bayesian networks**

Semantics-based:

- **Ordered Binary Decision Diagrams** (OBDDs)/ **Deterministic Structured Decomposable Negation Normal Forms** (d-SDNNFs)
  - → #SAT and probabilistic evaluation are easy because these classes have strong semantic constraints
    - Used to understand **#SAT solvers**

Target classes in knowledge compilation

For #SAT and probabilistic evaluation, two main restrictions on compilation targets:

Width-based:

- Bounded **pathwidth**/**treewidth** Boolean circuits, CNFs, DNFs, etc.
  → message passing algorithm for #SAT and probabilistic evaluation
    - Links with **Bayesian networks**

Semantics-based:

- **Ordered Binary Decision Diagrams** (OBDDs)/ **Deterministic Structured Decomposable Negation Normal Forms** (d-SDNNFs)
  → #SAT and probabilistic evaluation are easy because these classes have strong semantic constraints
    - Used to understand **#SAT solvers**

Question: what are the links beetween the two?

Plan

- DNF/CNF φ of **pathwidth** $\leqslant k$ $\xrightarrow{O(|\varphi| \times exp(k))}$ **OBDD** (not us)

Plan

- DNF/CNF φ of **pathwidth** $\leqslant k$ $\xrightarrow{O(|\varphi| \times exp(k))}$ **OBDD** (not us)
- + matching lower bound

Plan

- DNF/CNF φ of **pathwidth** $\leqslant k$ $\xrightarrow{O(|\varphi| \times exp(k))}$ **OBDD** (not us)
- + matching lower bound

Then

- Circuit C of **treewidth** $\leqslant k$ $\xrightarrow{O(|C| \times \exp(k))}$ **d-SDNNF**

Plan

- DNF/CNF φ of **pathwidth** $\leqslant k$ $\xrightarrow{O(|\varphi| \times exp(k))}$ **OBDD** (not us)
- + matching lower bound

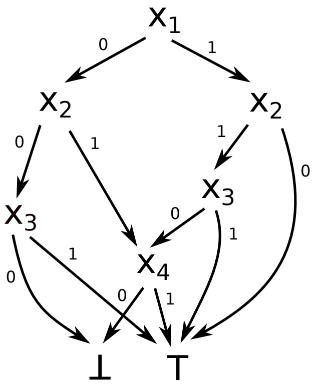
Then

- Circuit C of **treewidth** $\leqslant k$ $\xrightarrow{O(|C| \times \exp(k))}$ **d-SDNNF**
- + $\simeq$ matching lower bound

Pathwidth and **OBDDs**

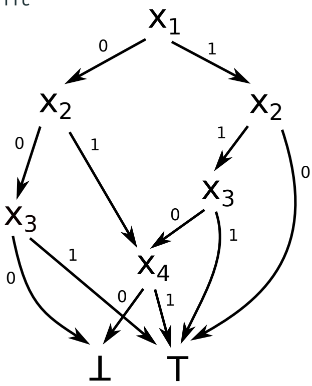
Ordered Binary Decision Diagrams (OBDDs)

- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables



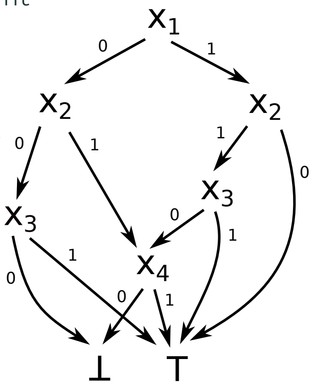
Ordered Binary Decision Diagrams (OBDDs)

- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables
- Semantics: follow the path of an assignment to get the value of the Boolean function



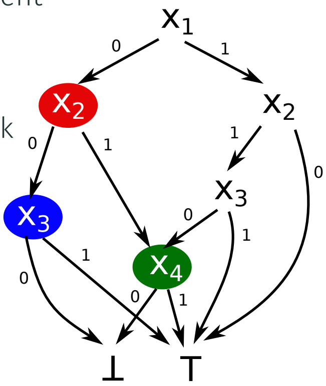
Ordered Binary Decision Diagrams (OBDDs)

- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables
- Semantics: follow the path of an assignment to get the value of the Boolean function
- There is a total order on the variables $\mathbf{v} = X_1\ X_2\ X_3\ X_4$ such that each root-to-sink path is compatible with $\mathbf{v}$



Ordered Binary Decision Diagrams (OBDDs)

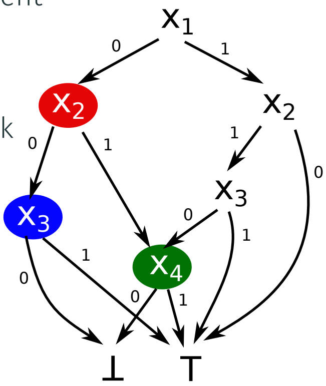
- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables
- Semantics: follow the path of an assignment to get the value of the Boolean function
- There is a total order on the variables $\mathbf{v} = X_1\ X_2\ X_3\ X_4$ such that each root-to-sink path is compatible with $\mathbf{v}$
- Compute probability bottom-up



Ordered Binary Decision Diagrams (OBDDs)

- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables
- Semantics: follow the path of an assignment to get the value of the Boolean function
- There is a total order on the variables $\mathbf{v} = X_1\ X_2\ X_3\ X_4$ such that each root-to-sink path is compatible with $\mathbf{v}$
- Compute probability bottom-up

$$\Pr_\pi(\color{red}{\bullet}) = \pi(X_2) \times \Pr_\pi(\color{green}{\bullet})$$
$$+(1 - \pi(X_2)) \times \Pr_\pi(\color{blue}{\bullet})$$

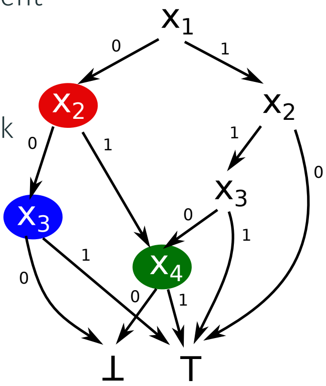


Ordered Binary Decision Diagrams (OBDDs)

- DAG with sink nodes $\{\top, \bot\}$ and internal nodes labeled by variables
- Semantics: follow the path of an assignment to get the value of the Boolean function
- There is a total order on the variables $\mathbf{v} = X_1\ X_2\ X_3\ X_4$ such that each root-to-sink path is compatible with $\mathbf{v}$
- Compute probability bottom-up

$$\Pr_\pi(\color{red}{\bullet}) = \pi(X_2) \times \Pr_\pi(\color{green}{\bullet}) + (1 - \pi(X_2)) \times \Pr_\pi(\color{blue}{\bullet})$$

- Width of the OBDD $\simeq$ largest number of nodes that are labeled by the same variable



Bounded pathwidth CNFs/DNFs

- **Pathwidth** of a DNF/CNF: that of its **Gaifman graph**
- Arity: size of the largest clause
- Degree: maximal number of clauses to which a variable belongs

Pathwidth and OBDDs: Upper and lower bounds

Upper bound:

Theorem (Bova & Slivovsky, 2017)

*Let φ be a CNF or DNF of **pathwidth** k. We can compile φ into an **OBDD** of width 2^{k+2} (hence of size $\leqslant nb\_vars \times 2^{k+2}$)*

Pathwidth and OBDDs: Upper and lower bounds

Upper bound:

Theorem (Bova & Slivovsky, 2017)

*Let φ be a CNF or DNF of **pathwidth** k. We can compile φ into an **OBDD** of width 2^{k+2} (hence of size $\leqslant$ nb_vars $\times\ 2^{k+2}$)*

Lower bound:

Theorem (Our contribution)

*Let φ be a **monotone** CNF or DNF of **pathwidth** k, and let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any **OBDD** for φ has width $\geqslant 2^{\left\lfloor \frac{k}{a^3 \times d^2} \right\rfloor}$*

Pathwidth and OBDDs: Upper and lower bounds

Upper bound:

Theorem (Bova & Slivovsky, 2017)

*Let φ be a CNF or DNF of **pathwidth** k. We can compile φ into an **OBDD** of width 2^{k+2} (hence of size $\leqslant$ nb_vars $\times 2^{k+2}$)*

Lower bound:

Theorem (Our contribution)

*Let φ be a **monotone** CNF or DNF of **pathwidth** k, and let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any **OBDD** for φ has width $\geqslant 2^{\left\lfloor \frac{k}{a^3 \times d^2} \right\rfloor}$*

- This is a generic lower bound

Pathwidth and OBDDs: Upper and lower bounds

Upper bound:

Theorem (Bova & Slivovsky, 2017)

*Let φ be a CNF or DNF of **pathwidth** k. We can compile φ into an **OBDD** of width 2^{k+2} (hence of size $\leqslant$ nb_vars $\times\ 2^{k+2}$)*

Lower bound:

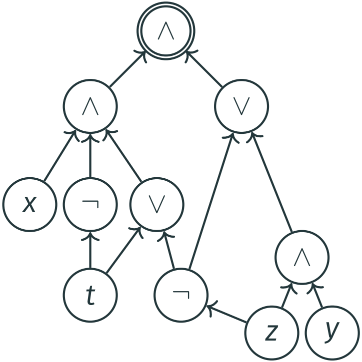
Theorem (Our contribution)

*Let φ be a **monotone** CNF or DNF of **pathwidth** k, and let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any **OBDD** for φ has width $\geqslant 2^{\left\lfloor \frac{k}{a^3 \times d^2} \right\rfloor}$*

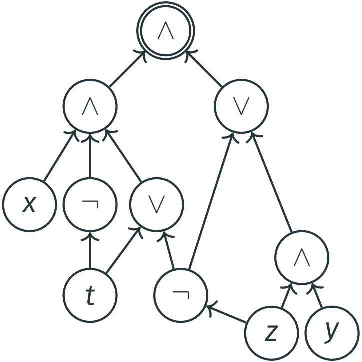
- This is a generic lower bound
- For monotone DNF/CNF φ of constant arity and degree, the smallest width of an OBDD for φ is $2^{\Theta(\text{pathwidth}(\varphi))}$

Treewidth and **d-SDNNFs**

Bounded treewidth Boolean circuits

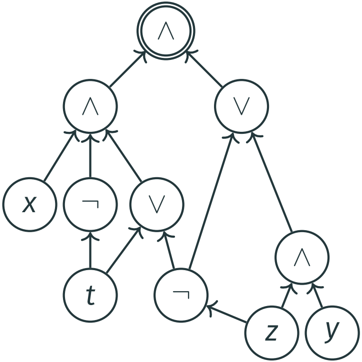


Bounded treewidth Boolean circuits



Treewidth of C = that of the underlying graph

Bounded treewidth Boolean circuits



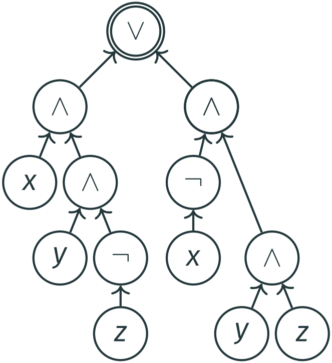
Treewidth of C = that of the underlying graph

We can do message passing:

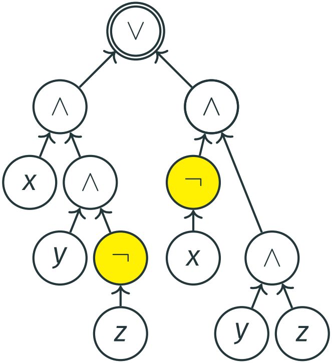
Theorem (Lauritzen & Spielgelhalter, 1988)

Fix $k \in \mathbb{N}$. Given a Boolean circuit C of treewidth $\leqslant k$, we can compute its probability in time $O(f(k) \times |C|)$, where f is singly exponential

d-SDNNF

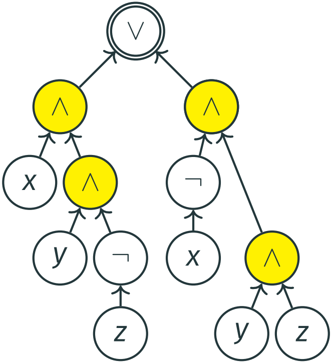


d-SDNNF



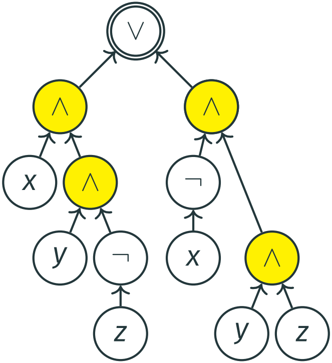
- Negation Normal Form: negations only applied to the leaves

d-SDNNF



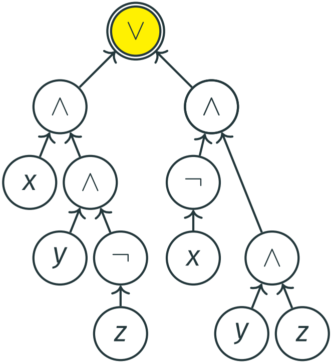
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)

d-SDNNF



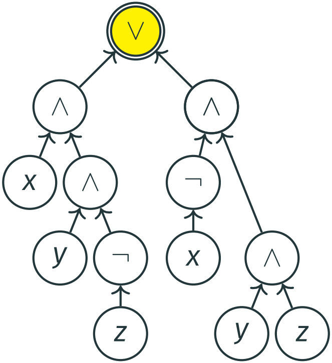
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)
  - → **SAT** can be solved efficiently

d-SDNNF



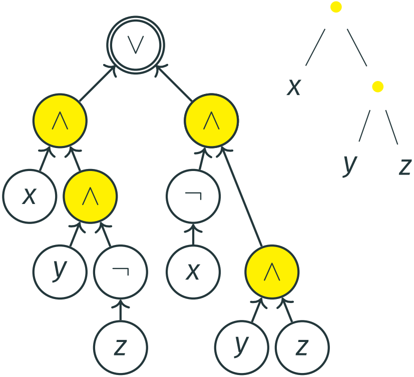
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of $\land$-gates are **independent** (no variable has a path to two different inputs of the same $\land$-gate)
  - $\rightarrow$ **SAT** can be solved efficiently
- Deterministic: inputs of $\lor$-gates are **mutually exclusive**

d-SDNNF



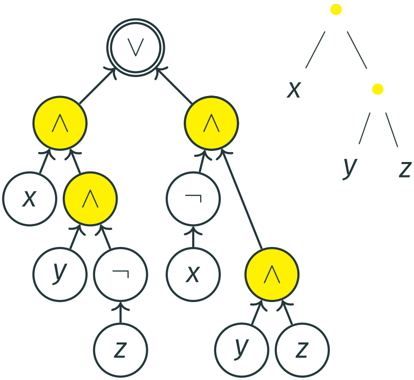
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)
  - → **SAT** can be solved efficiently
- Deterministic: inputs of ∨-gates are **mutually exclusive**
  - → **#SAT** and **probability evaluation**

d-SDNNF



- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)
  - → **SAT** can be solved efficiently
- Deterministic: inputs of ∨-gates are **mutually exclusive**
  - → **#SAT** and **probability evaluation**
- Structured: there is a **vtree** that structures the ∧-gates

d-SDNNF



- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)
  - → **SAT** can be solved efficiently
- Deterministic: inputs of ∨-gates are **mutually exclusive**
  - → **#SAT** and **probability evaluation**
- Structured: there is a **vtree** that structures the ∧-gates
  - → **Enumeration**

Treewidth and d-SDNNFs: Upper bound

Theorem (Bova & Szeider, 2017)

*Let C be a Boolean circuit on m variables of **treewidth** $\leqslant k$.*
***There exists** a **d-SDNNF** equivalent to C of size $O(m \times g(k))$,*
*where g is **doubly** exponential*

Treewidth and d-SDNNFs: Upper bound

Theorem (Bova & Szeider, 2017)

*Let C be a Boolean circuit on m variables of **treewidth** $\leqslant k$.*
***There exists** a **d-SDNNF** equivalent to C of size $O(m \times g(k))$,*
*where g is **doubly** exponential*

Drawbacks: non constructive

Treewidth and d-SDNNFs: Upper bound

Theorem (Bova & Szeider, 2017)

Let C be a Boolean circuit on m variables of ***treewidth*** $\leqslant k$.
There exists *a* ***d-SDNNF*** *equivalent to C of size $O(m \times g(k))$,*
where g is ***doubly*** *exponential*

Drawbacks: non constructive

Theorem (Our contribution)

Let C be a Boolean circuit of ***treewidth*** $\leqslant k$.
We can compute *a* ***d-SDNNF*** *equivalent to C in time $O(|C| \times f(k))$,*
where f is ***singly*** *exponential*

Treewidth and d-SDNNFs: Upper bound

Theorem (Bova & Szeider, 2017)

Let C be a Boolean circuit on m variables of ***treewidth*** $\leqslant k$.
There exists *a* ***d-SDNNF*** *equivalent to C of size $O(m \times g(k))$,*
where g is ***doubly*** *exponential*

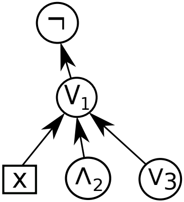
Drawbacks: non constructive

Theorem (Our contribution)

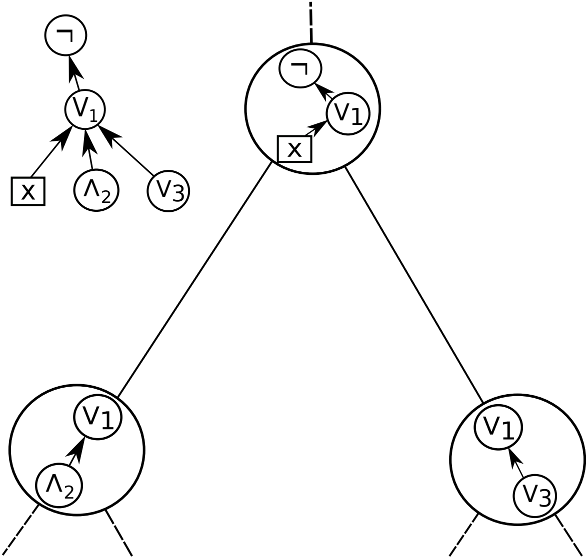
Let C be a Boolean circuit of ***treewidth*** $\leqslant k$.
We can compute *a* ***d-SDNNF*** *equivalent to C in time $O(|C| \times f(k))$,*
where f is ***singly*** *exponential*

Applications: recapturing message passing, and enumeration of satisfying valuations

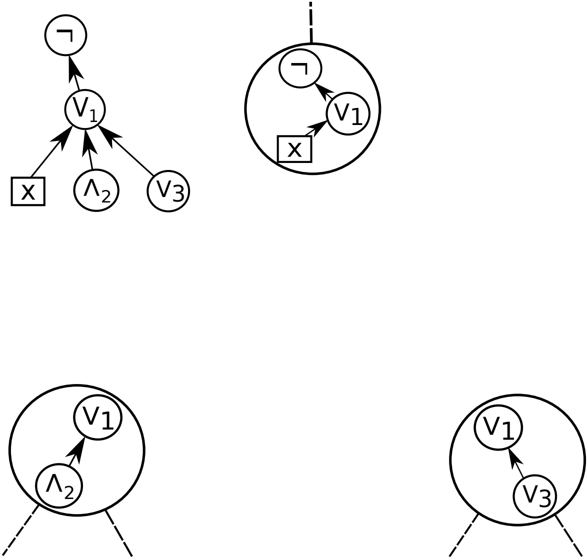
Construction sketch



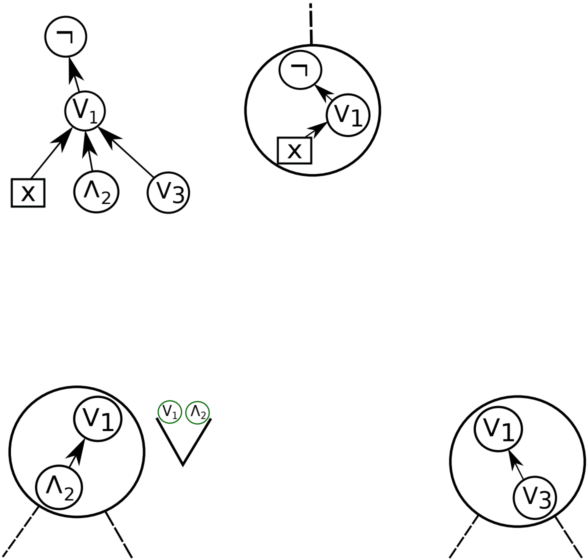
Construction sketch



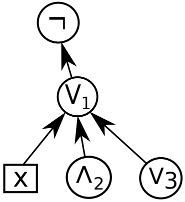
Construction sketch

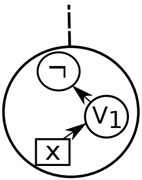


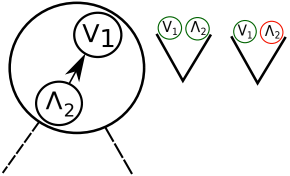
Construction sketch

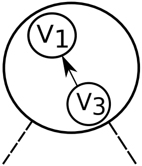


Construction sketch

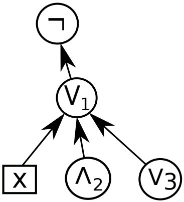


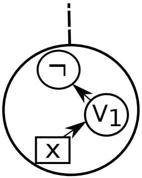


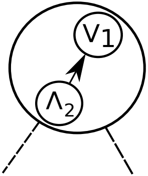


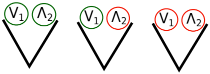


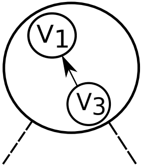
Construction sketch



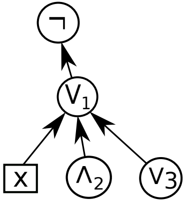


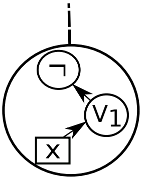


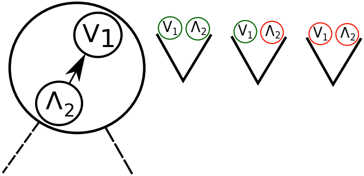


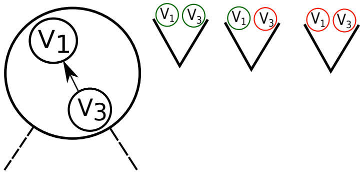


Construction sketch

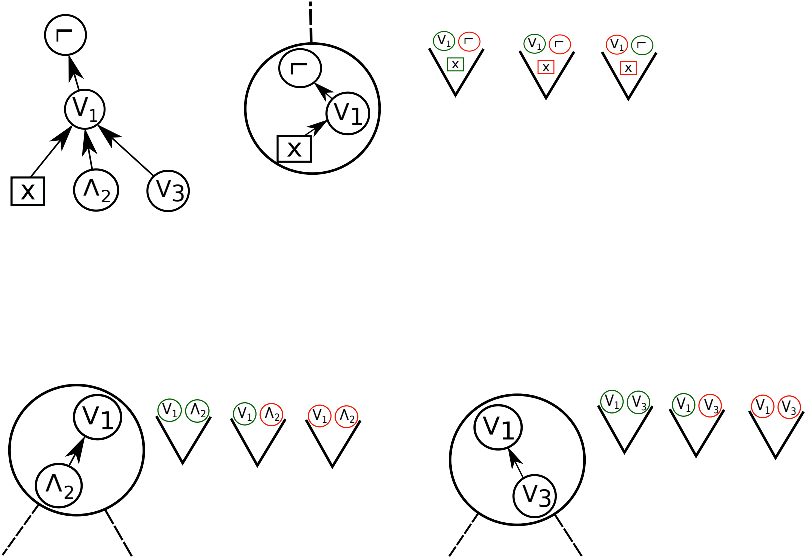




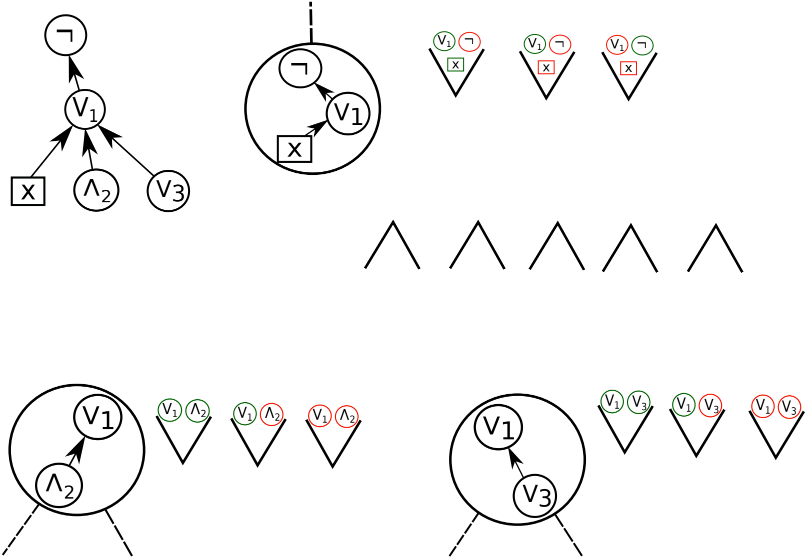




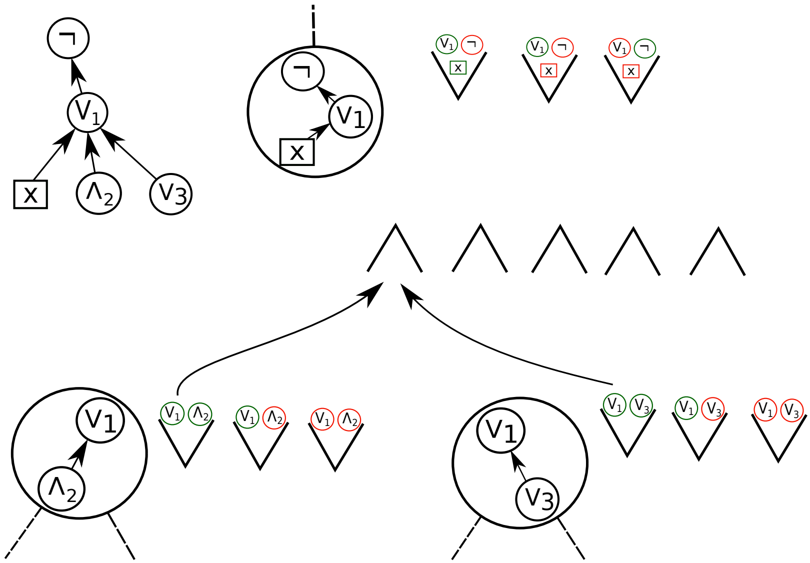
Construction sketch



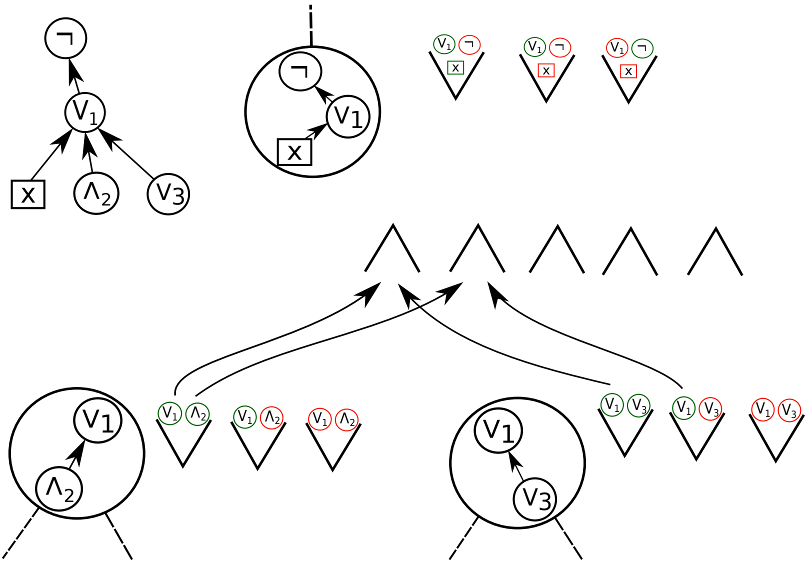
Construction sketch



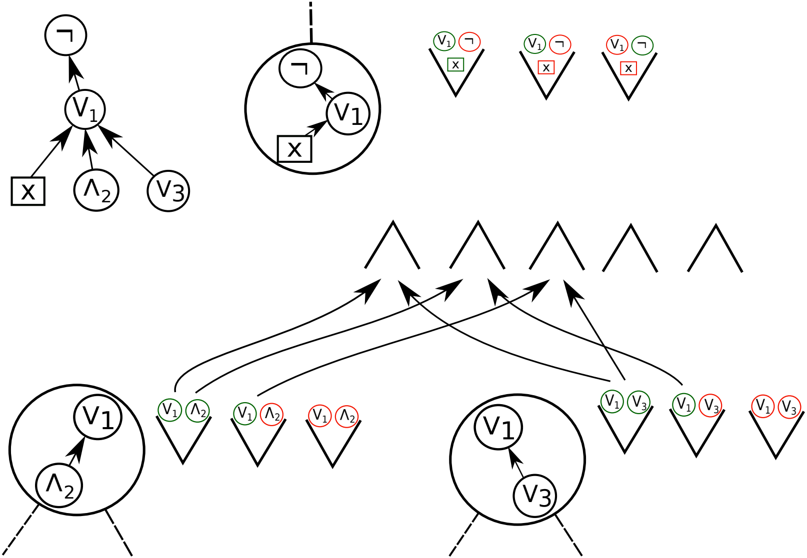
Construction sketch



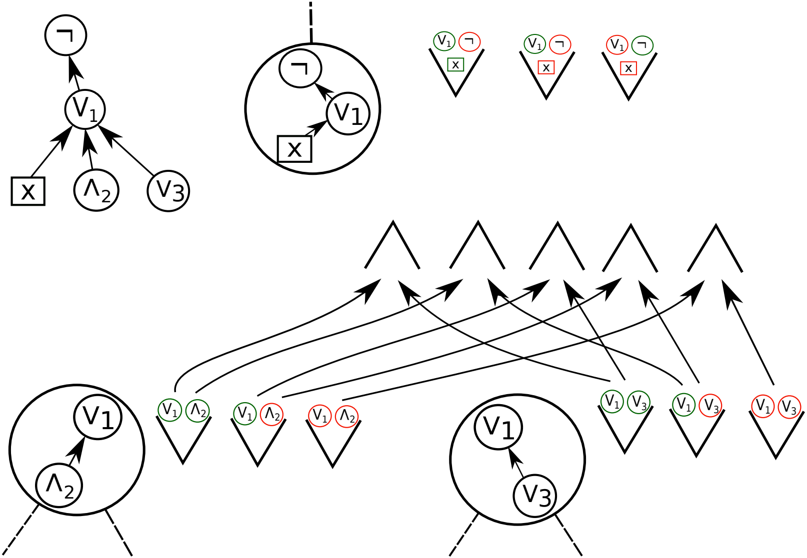
Construction sketch



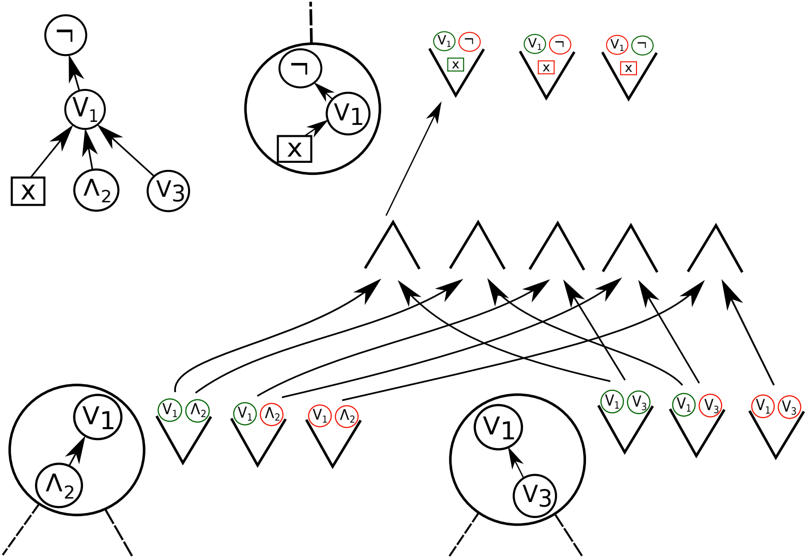
Construction sketch



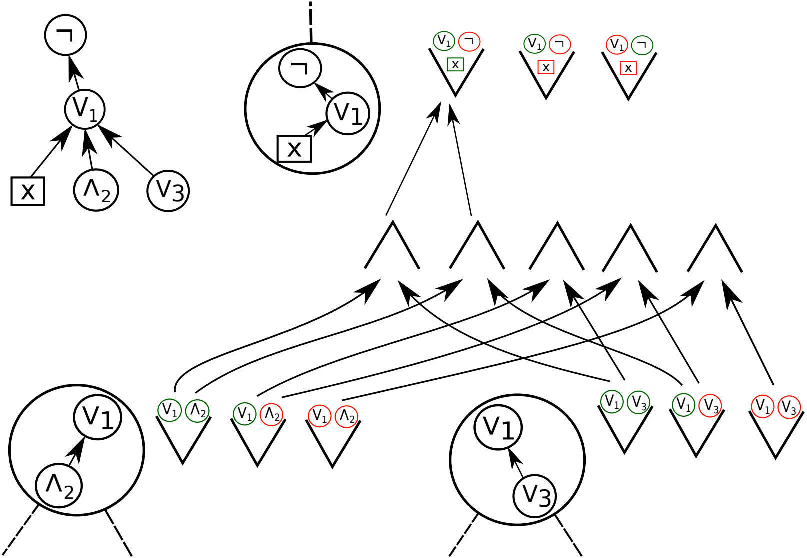
Construction sketch



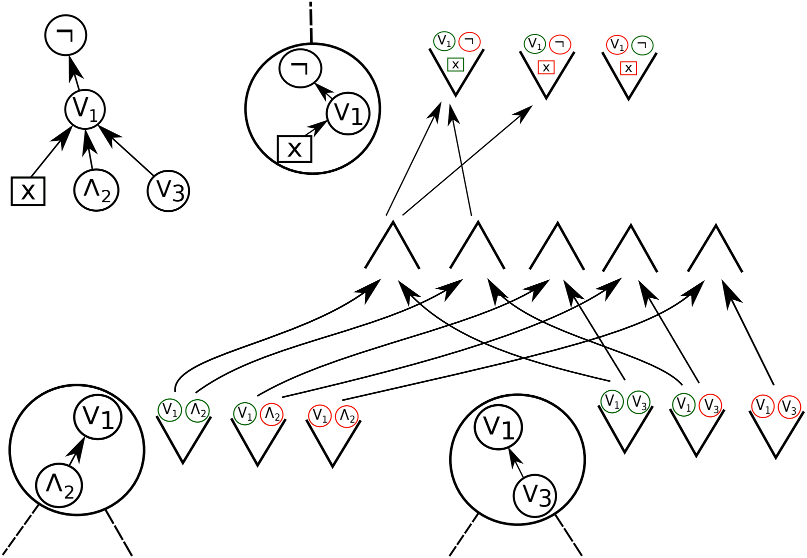
Construction sketch



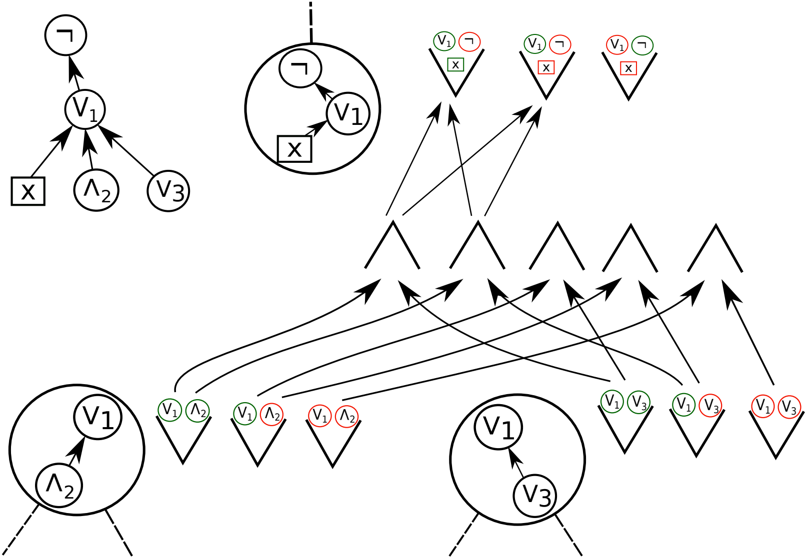
Construction sketch



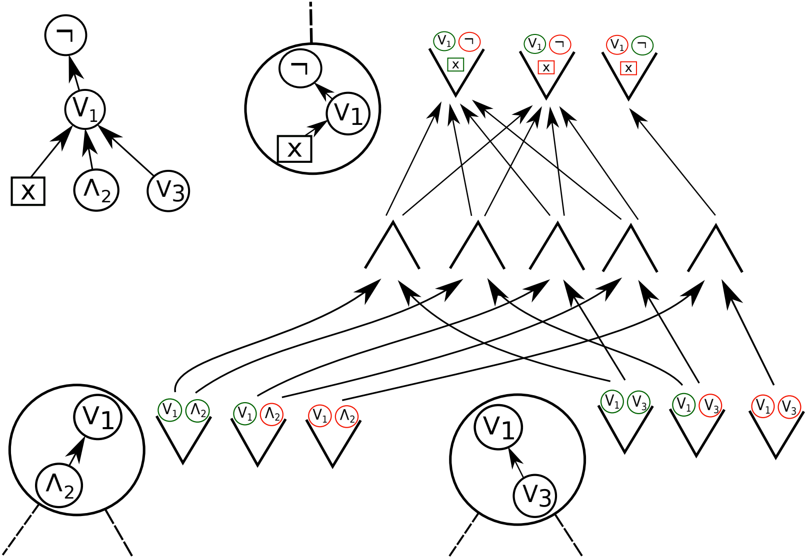
Construction sketch



Construction sketch



Construction sketch



Treewidth and d-SDNNFs: Lower bound

Theorem

*Let φ be a **monotone DNF** of **treewidth** k, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any **d-SDNNF** for φ has size* $\geqslant 2^{\left\lfloor \frac{k}{3 \times a^3 \times d^2} \right\rfloor} - 1$

Treewidth and d-SDNNFs: Lower bound

Theorem

Let φ be a ***monotone DNF*** *of* ***treewidth*** *k, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any* ***d-SDNNF*** *for φ has size* $\geqslant 2^{\left\lfloor \frac{k}{3 \times a^3 \times d^2} \right\rfloor} - 1$

- For **CNFs**, the bound even works for (non-deterministic) **SDNNF**

Treewidth and d-SDNNFs: Lower bound

Theorem

*Let φ be a **monotone DNF** of **treewidth** k, let $a := \mathrm{arity}(\varphi)$ and $d := \mathrm{degree}(\varphi)$. Then any **d-SDNNF** for φ has size* $\geqslant 2^{\left\lfloor \frac{k}{3 \times a^3 \times d^2} \right\rfloor} - 1$

- For **CNFs**, the bound even works for (non-deterministic) **SDNNF**
- Again, the bound is generic: it applies to *any* monotone DNF/CNF

Proof Sketch for CNFs (1/2)

Use the connection made in [Bova, Capelli & Mengel, 2016] between the notion of combinatorial rectangle in **communication complexity** and SDNNFs.

Definition

A (X, Y)-rectangle is a Boolean function $R : 2^{X \cup Y} \to \{0, 1\}$ that can be written as $R_X \land R_Y$, for some Boolean functions $R_X : 2^X \to \{0, 1\}$ and $R_Y : 2^Y \to \{0, 1\}$. A (X, Y)-rectangle cover of a function $f : 2^{X \cup Y} \to \{0, 1\}$ is a set $\{R_1, \cdots, R_n\}$ of (X, Y)-rectangles such that $f \equiv \bigvee_{i=1}^{n} R_i$.

Proof Sketch for CNFs (1/2)

Use the connection made in [Bova, Capelli & Mengel, 2016] between the notion of combinatorial rectangle in **communication complexity** and SDNNFs.

Definition

A (X, Y)-rectangle is a Boolean function $R : 2^{X \cup Y} \to \{0, 1\}$ that can be written as $R_X \wedge R_Y$, for some Boolean functions $R_X : 2^X \to \{0, 1\}$ and $R_Y : 2^Y \to \{0, 1\}$. A (X, Y)-rectangle cover of a function $f : 2^{X \cup Y} \to \{0, 1\}$ is a set $\{R_1, \cdots, R_n\}$ of (X, Y)-rectangles such that $f \equiv \bigvee_{i=1}^{n} R_i$.

Theorem (Bova, Capelli & Mengel, 2016)

Let C be an SDNNF computing a function φ on variables V, structured by a v-tree T. Let $n \in T$, and let (X, Y) be the partition of V that n induces. Then φ has a (X, Y)-rectangle cover of size at most $|C|$.

Proof Sketch for CNFs (2/2)

A CNF having no small rectangle cover:

Theorem (Sherstov, 2014)
Let $X = \{x_1, \ldots, x_n\}$ *and* $Y = \{y_1, \ldots, y_n\}$ *be two disjoint sets of variables. Then any* (X, Y)*-rectangle cover of the Boolean function* $\mathrm{SCOV}_n(X, Y) := \bigwedge_{i=1}^{n} x_i \vee y_i$ *has size* $\geqslant 2^n$.

Proof Sketch for CNFs (2/2)

A CNF having no small rectangle cover:

Theorem (Sherstov, 2014)

Let $X = \{x_1, \ldots, x_n\}$ *and* $Y = \{y_1, \ldots, y_n\}$ *be two disjoint sets of variables. Then any* (X, Y)*-rectangle cover of the Boolean function* $\mathrm{SCOV}_n(X, Y) := \bigwedge_{i=1}^{n} x_i \vee y_i$ *has size* $\geqslant 2^n$.

We show that we can find $\mathrm{SCOV}_{\frac{k}{3 \times a^3 \times d^2}}(X, Y)$ within any CNF of treewidth $\geqslant k$.

Proof Sketch for CNFs (2/2)

A CNF having no small rectangle cover:

Theorem (Sherstov, 2014)

Let $X = \{x_1, \ldots, x_n\}$ *and* $Y = \{y_1, \ldots, y_n\}$ *be two disjoint sets of variables. Then any* (X, Y)*-rectangle cover of the Boolean function* $\mathrm{SCOV}_n(X, Y) := \bigwedge_{i=1}^{n} x_i \vee y_i$ *has size* $\geqslant 2^n$.

We show that we can find $\mathrm{SCOV}_{\frac{k}{3 \times a^3 \times d^2}}(X, Y)$ within any CNF of treewidth $\geqslant k$.

→ Rephrase treewidth as treesplitwidth, a new measure capturing the 'performance' of a v-tree

Conclusion

- Strong connections between width- and semantics-based restrictions in knowledge compilation:

Conclusion

- Strong connections between width- and semantics-based restrictions in knowledge compilation:
  - $\rightarrow$ Recapture message passing on bounded **treewidth** circuits by compiling them to **d-SDNNF**

Conclusion

- Strong connections between width- and semantics-based restrictions in knowledge compilation:
  - $\rightarrow$ Recapture message passing on bounded **treewidth** circuits by compiling them to **d-SDNNF**
  - $\rightarrow$ Compilation is singly exponential in the **treewidth** of the circuit and cannot be avoided

Conclusion

- Strong connections between width- and semantics-based restrictions in knowledge compilation:
  - → Recapture message passing on bounded **treewidth** circuits by compiling them to **d-SDNNF**
  - → Compilation is singly exponential in the **treewidth** of the circuit and cannot be avoided
  - → The width of an **OBDD** and the **pathwidth** of a DNF/CNF are within a constant of each other

Conclusion

- Strong connections between width- and semantics-based restrictions in knowledge compilation:
  - → Recapture message passing on bounded **treewidth** circuits by compiling them to **d-SDNNF**
  - → Compilation is singly exponential in the **treewidth** of the circuit and cannot be avoided
  - → The width of an **OBDD** and the **pathwidth** of a DNF/CNF are within a constant of each other
- Future work:
  - → Get rid of arity and degree assumptions?
  - → Notion of width for d-SDNNFs?
  - → Lower bound for d-DNNFs?

Thanks for your attention!