Towards Characterizing Efficient Boolean Functional Synthesis

Supratik Chakraborty

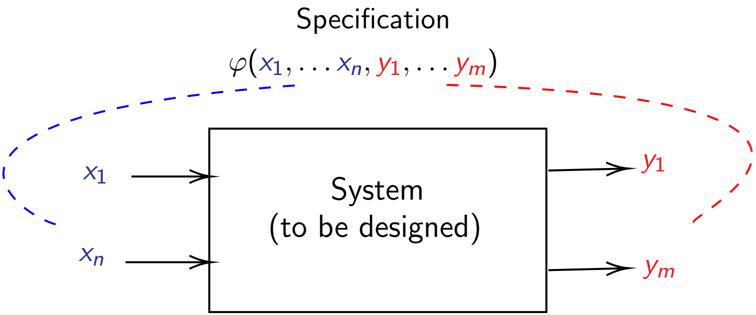
Indian Institute of Technology Bombay

Joint work with S. Akshay, Jatin Arora, Aman Bansal, Divya Raghunathan, Preey Shah, Shetal Shah, Krishna S.

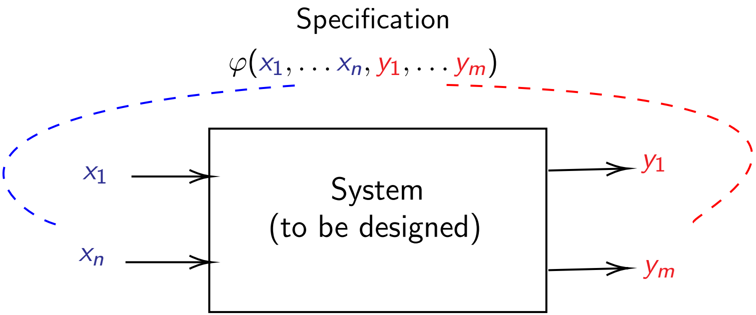
Several interesting discussions with Kuldeep Meel and Dror Fried

SAT: Theory, Practice and Beyond Reunion Workshop, June 2022

Synthesis: Recap from previous talk

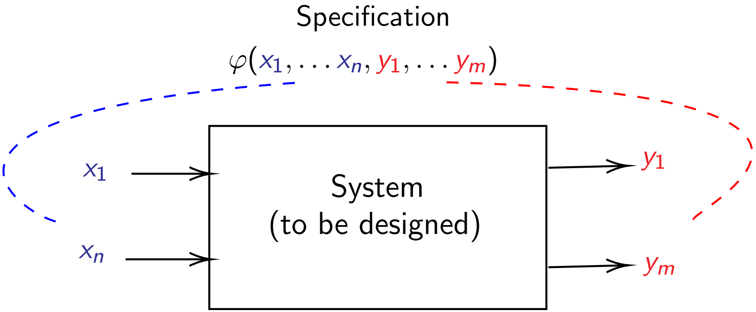


Synthesis: Recap from previous talk



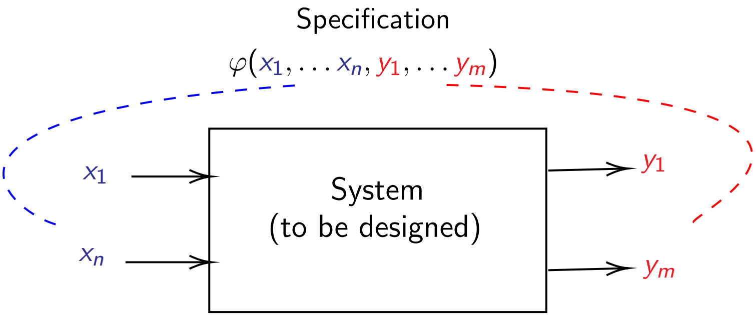
- Goal: Automatically **synthesize system** s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$
  - x_j *input* variables (vector X)
  - y_k *output* variables (vector Y)

Synthesis: Recap from previous talk



- Goal: Automatically **synthesize system** s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ **whenever possible**.
  - x_j *input* variables (vector X)
  - y_k *output* variables (vector Y)

Synthesis: Recap from previous talk



- Goal: Automatically **synthesize system** s.t. it satisfies $\varphi(x_1, .., x_n, y_1, .., y_m)$ **whenever possible**.
  - x_j *input* variables (vector X)
  - y_k *output* variables (vector Y)
- Express $y_1, \ldots y_m$ as $F_1(\mathsf{X}), \ldots F_m(\mathsf{X})$ s.t. $\varphi(\mathsf{X}, \mathsf{F}(\mathsf{X}))$ is satisfied.

Boolean Functional Synthesis (BFnS)

Formal definition

Given Boolean relation $\varphi(x_1, .., x_n, y_1, .., y_m)$

- x_k *input* variables (vector X)
- y_j *output* variables (vector Y)

Boolean Functional Synthesis (BFnS)

Formal definition

Given Boolean relation $\varphi(x_1, .., x_n, y_1, .., y_m)$

- x_k *input* variables (vector X)
- y_j *output* variables (vector Y)

Synthesize Boolean functions $F_j(\mathrm{X})$ for each y_j s.t.

$$\forall \mathrm{X}(\ \exists y_1 \dots y_m\ \varphi(\mathrm{X}, y_1 \dots y_m) \ \Leftrightarrow\ \varphi(\mathrm{X}, F_1(\mathrm{X}), \dots F_m(\mathrm{X}))\ )$$

Boolean Functional Synthesis (BFnS)

Formal definition

Given Boolean relation $\varphi(x_1, .., x_n, y_1, .., y_m)$

- x_k *input* variables (vector X)
- y_j *output* variables (vector Y)

Synthesize Boolean functions $F_j(X)$ for each y_j s.t.

$$\forall X(\ \exists y_1 \ldots y_m \ \varphi(X, y_1 \ldots y_m) \ \Leftrightarrow \ \varphi(X, F_1(X), \ldots F_m(X)) \ )$$

$F_j(X)$ is also called a *Skolem function* for y_j in φ.

How Hard is Boolean Skolem Function Synthesis?

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard (not surprising!)

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard (not surprising!)

Space complexity [Akshay+'18]

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard (not surprising!)

Space complexity [Akshay+'18]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist $\varphi(X, Y)$ for which Skolem function sizes are super-polynomial in $|\varphi|$.

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard (not surprising!)

Space complexity [Akshay+'18]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist $\varphi(X, Y)$ for which Skolem function sizes are super-polynomial in $|\varphi|$.
- Unless non-uniform exponential-time hypothesis fails, there exist $\varphi(X, Y)$ for which Sk. fn. sizes are exponential in $|\varphi|$.

How Hard is Boolean Skolem Function Synthesis?

Representation

Spec $\varphi(X, Y)$ & Skolem functions $F(X)$: NNF Boolean circuits

Time complexity

Boolean Functional Synthesis is *NP*-hard (not surprising!)

Space complexity [Akshay+'18]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist $\varphi(X, Y)$ for which Skolem function sizes are super-polynomial in $|\varphi|$.
- Unless non-uniform exponential-time hypothesis fails, there exist $\varphi(X, Y)$ for which Sk. fn. sizes are exponential in $|\varphi|$.

Silver lining: [Akshay+'18,'19]

- Solvable in polynomial (in $|\varphi|$) time and space if φ is represented in special normal forms

Synthesis Basics

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$:

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(X, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs X, output y_m

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(X, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs X, output y_m
- Synthesize $F_m(X)$ for y_m from φ'

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(X, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs X, output y_m
- Synthesize $F_m(X)$ for y_m from φ'
- Construct *new spec* $\varphi''(X, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\ \varphi$
  - Inputs X, y_m; output y_{m-1}

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(X, y_m) \equiv \exists y_1 \ldots y_{m-1}\, \varphi$
  - Inputs X, output y_m
- Synthesize $F_m(X)$ for y_m from φ'
- Construct *new spec* $\varphi''(X, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\, \varphi$
  - Inputs X, y_m; output y_{m-1}
- Synthesize $F_{m-1}(X, y_m)$ for y_{m-1}; substitute $F_m(X)$ for y_m

Synthesis Basics

1-output synthesis: easy!

Spec $\varphi(X, y_1)$: $\neg\varphi(X, 0)$ and $\varphi(X, 1)$ suffice for $F_1(X)$

Multi-output synthesis

Spec $\varphi(X, y_1, \ldots y_m)$: Transform to 1-output synthesis

- Construct *new spec* $\varphi'(X, y_m) \equiv \exists y_1 \ldots y_{m-1}\ \varphi$
  - Inputs X, output y_m
- Synthesize $F_m(X)$ for y_m from φ'
- Construct *new spec* $\varphi''(X, y_{m-1}, y_m) \equiv \exists y_1 \ldots y_{m-2}\ \varphi$
  - Inputs X, y_m; output y_{m-1}
- Synthesize $F_{m-1}(X, y_m)$ for y_{m-1}; substitute $F_m(X)$ for y_m
- Repeat ...

Why Do Some Representations Help?

Efficient computation of
$\exists y_1 \dots y_i \; \varphi(X, y_1, \dots y_m)$
$\forall i \in \{1, \dots m\}$

$\updownarrow$

Efficient computation of
Sk fn $F_i(X)$ in $\varphi(X, y_1, \dots y_m)$
$\forall i \in \{1, \dots m\}$

Why Do Some Representations Help?

Efficient computation of
$\exists y_1 \ldots y_i \; \varphi(X, y_1, \ldots y_m)$
$\forall i \in \{1, \ldots m\}$

If a representation enables this

$\updownarrow$

Efficient computation of
Sk fn $F_i(X)$ in $\varphi(X, y_1, \ldots y_m)$
$\forall i \in \{1, \ldots m\}$

Why Do Some Representations Help?

Efficient computation of
$\exists y_1 \ldots y_i\ \varphi(X, y_1, \ldots y_m)$
$\forall i \in \{1, \ldots m\}$

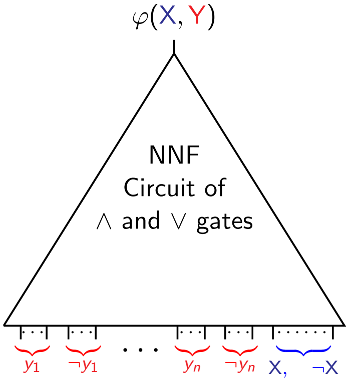
If a representation enables this

$\updownarrow$

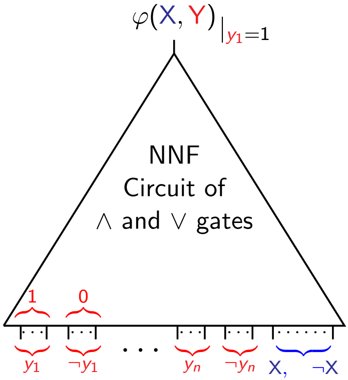
Efficient computation of
Sk fn $F_i(X)$ in $\varphi(X, y_1, \ldots y_m)$
$\forall i \in \{1, \ldots m\}$

then it also enables this

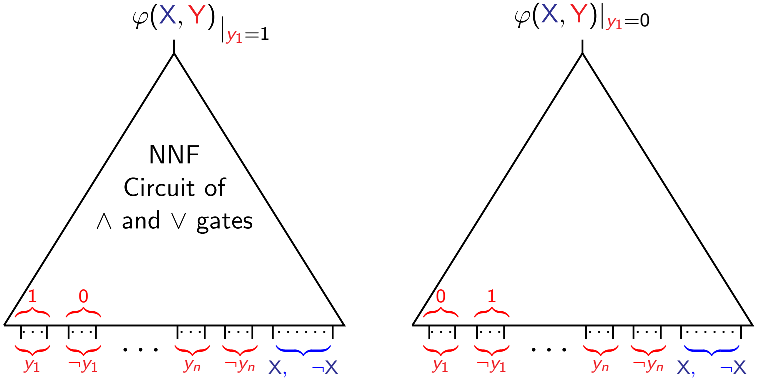
Existential Quantification with NNF circuits



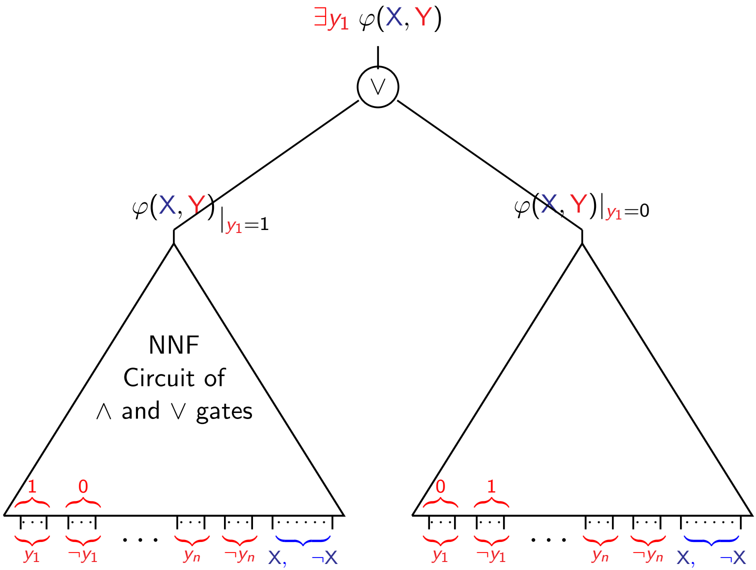
Existential Quantification with NNF circuits



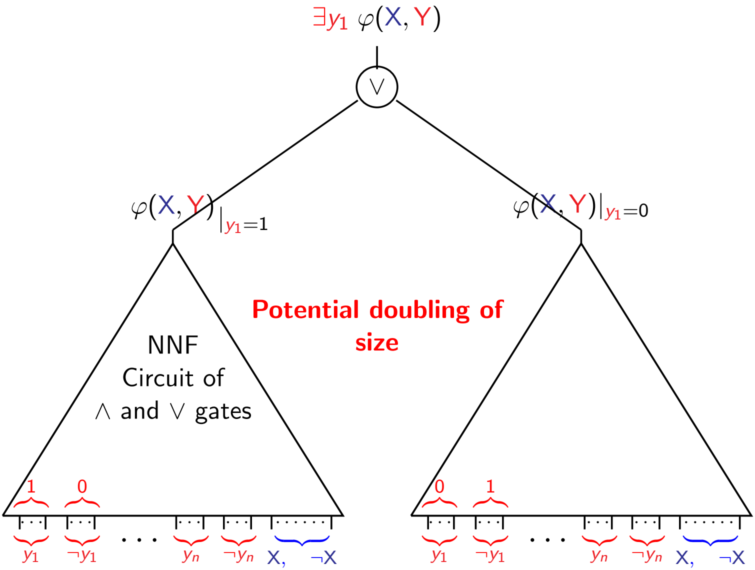
Existential Quantification with NNF circuits



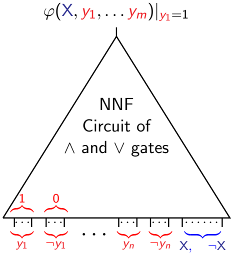
Existential Quantification with NNF circuits

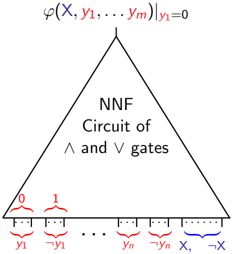


Existential Quantification with NNF circuits

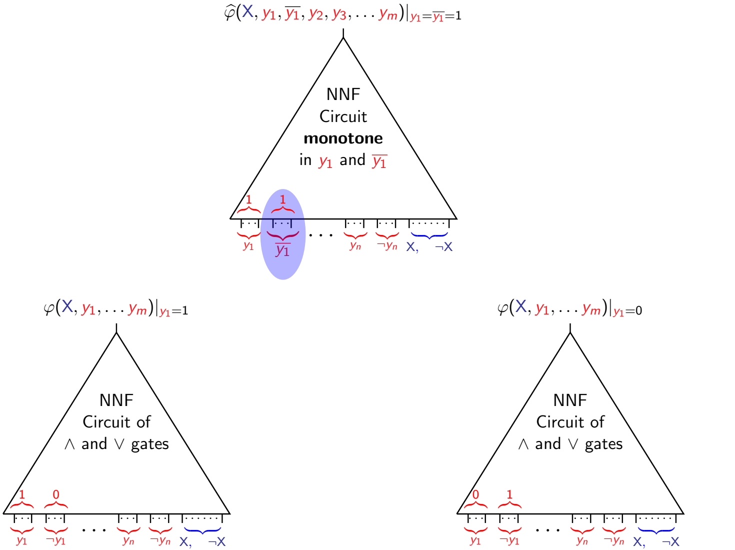


Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling

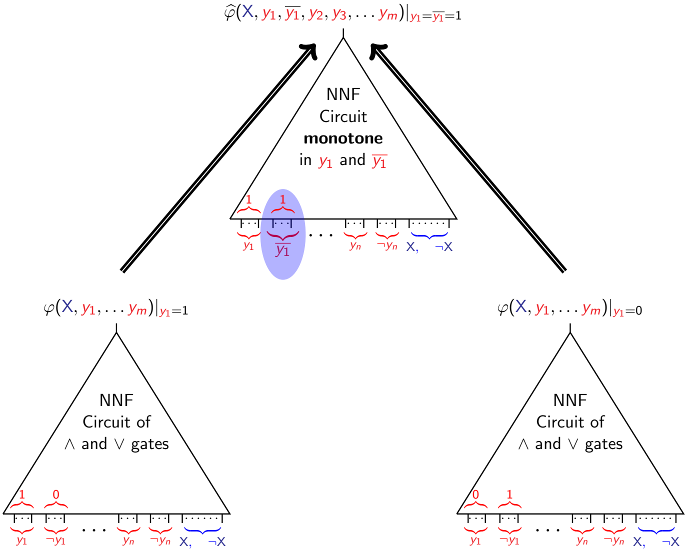




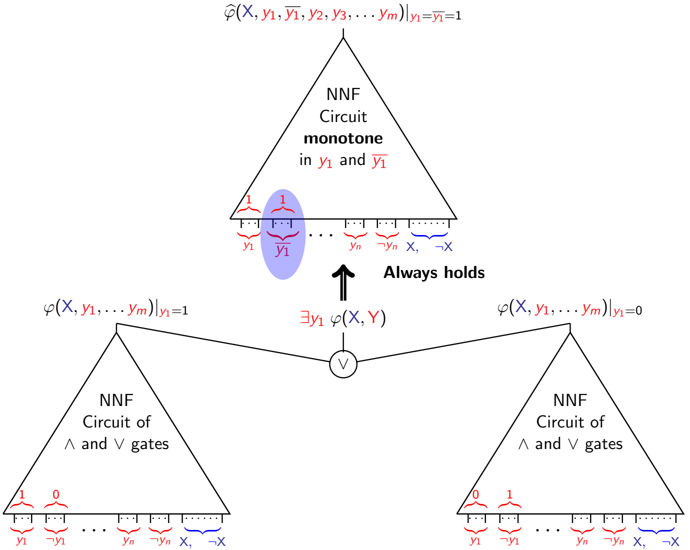
Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling



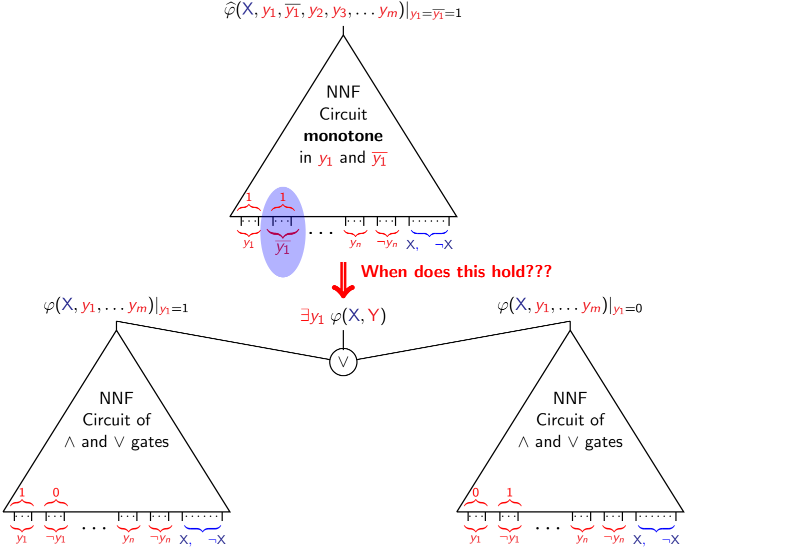
Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling



Over-approximating $\exists y_1\ \varphi(X, Y)$ Sans Doubling



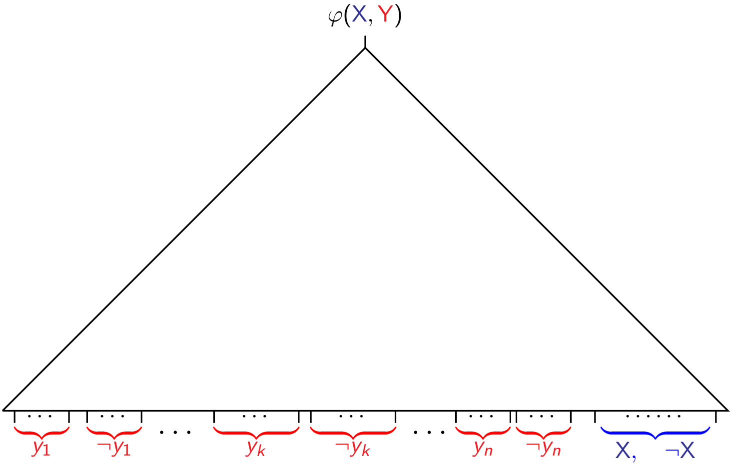
Can We Represent Quantification **Exactly** sans Blow-up?



Special Normal Forms

Decomposable Negation Normal Form (DNNF): Forbidden **structure**

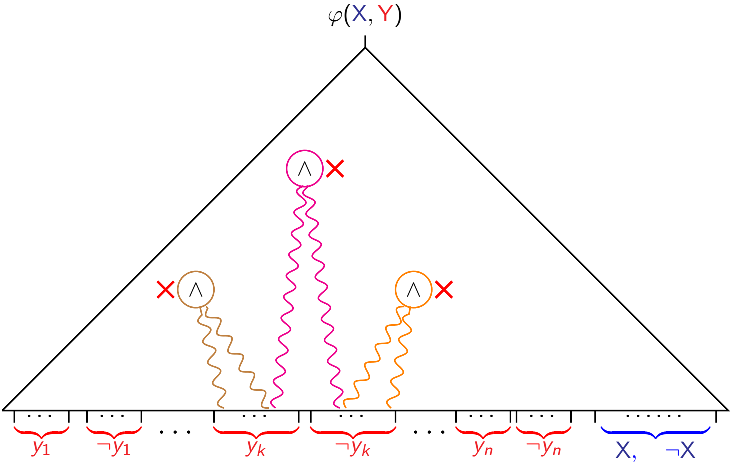
[Darwiche, JACM 2001]



Special Normal Forms

Decomposable Negation Normal Form (DNNF): Forbidden **structure**

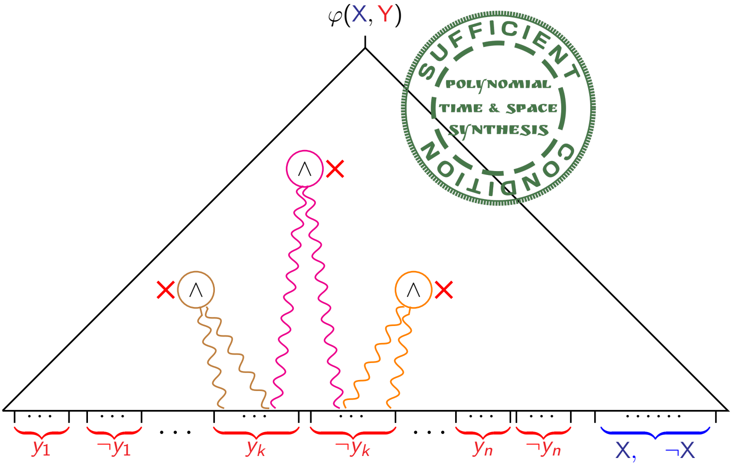
[Darwiche, JACM 2001]



Special Normal Forms

Decomposable Negation Normal Form (DNNF): Forbidden **structure**

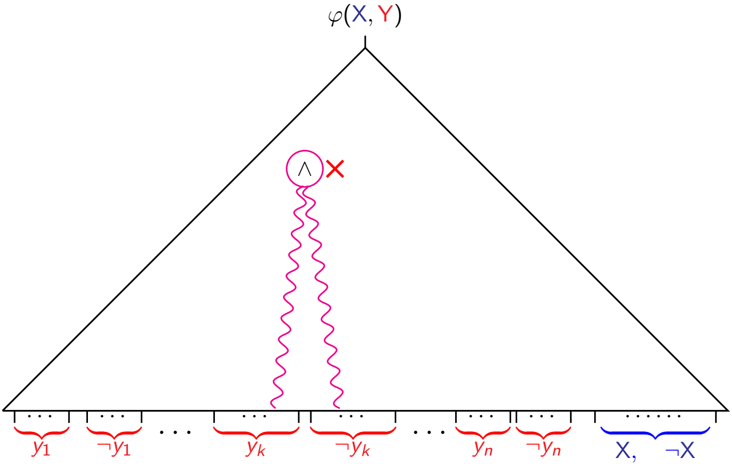
[Darwiche, JACM 2001]



Special Normal Forms

Weak DNNF (wDNNF): Forbidden **structure**

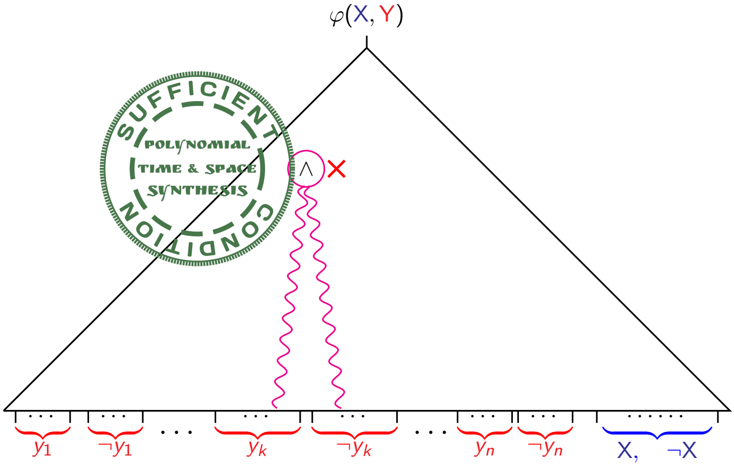
[Akshay,Chakraborty,Goel,Kulal,Shah CAV18, FMSD20]



Special Normal Forms

Weak DNNF (wDNNF): Forbidden **structure**

[Akshay,Chakraborty,Goel,Kulal,Shah CAV18, FMSD20]

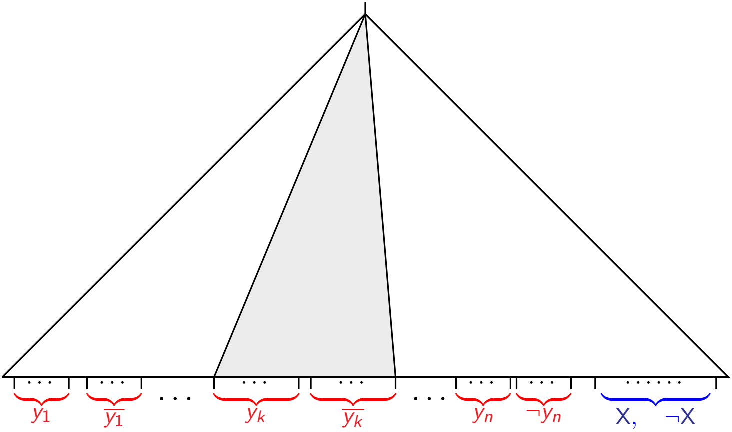


Special Normal Forms

Synthesis Negation Normal Form (SynNNF): Forbidden **semantics**

[Akshay,Arora,Chakraborty,Krishna,Raghunathan,Shah FMCAD19]

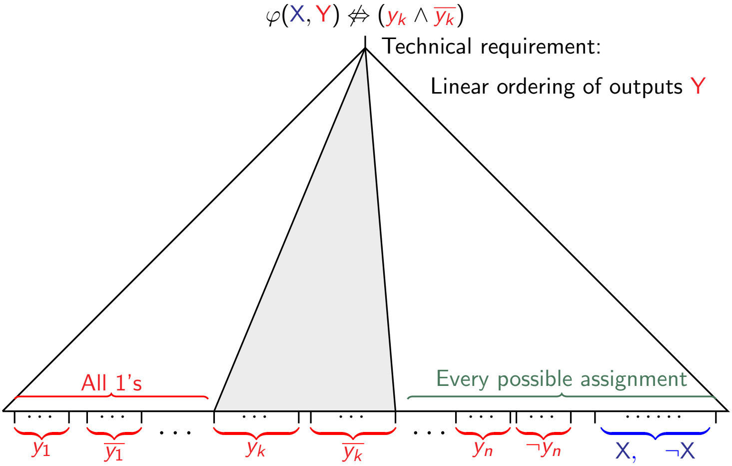
$\varphi(X, Y) \not\Leftarrow (y_k \wedge \overline{y_k})$



Special Normal Forms

Synthesis Negation Normal Form (SynNNF): Forbidden **semantics**

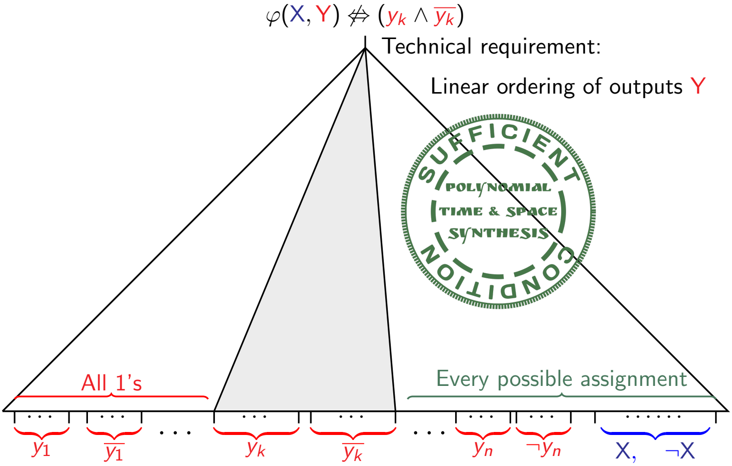
[Akshay,Arora,Chakraborty,Krishna,Raghunathan,Shah FMCAD19]



Special Normal Forms

Synthesis Negation Normal Form (SynNNF): Forbidden **semantics**

[Akshay,Arora,Chakraborty,Krishna,Raghunathan,Shah FMCAD19]



Can we get necessary & sufficient condition?

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

P1 : BFnS is poly-time for $\mathcal{C}^\star$

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

P1 : BFnS is poly-time for $\mathcal{C}^\star$

P2 : For every class $\mathcal{C}$ of ckts:

1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

P1 : BFnS is poly-time for $\mathcal{C}^\star$

P2 : For every class $\mathcal{C}$ of ckts:

1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.
2. BFnS is poly-size for $\mathcal{C}$ iff $\mathcal{C}$ compiles to poly-size ckts in $\mathcal{C}^\star$

Can we get necessary & sufficient condition?

Characterizing poly-time and poly-size BFnS

Does there exist a "semantically universal" class $\mathcal{C}^\star$ of ckts s.t.:

- P1 : BFnS is poly-time for $\mathcal{C}^\star$
- P2 : For every class $\mathcal{C}$ of ckts:
  1. BFnS is poly-time for $\mathcal{C}$ iff $\mathcal{C}$ compiles to $\mathcal{C}^\star$ in poly-time.
  2. BFnS is poly-size for $\mathcal{C}$ iff $\mathcal{C}$ compiles to poly-size ckts in $\mathcal{C}^\star$

Our Main Result

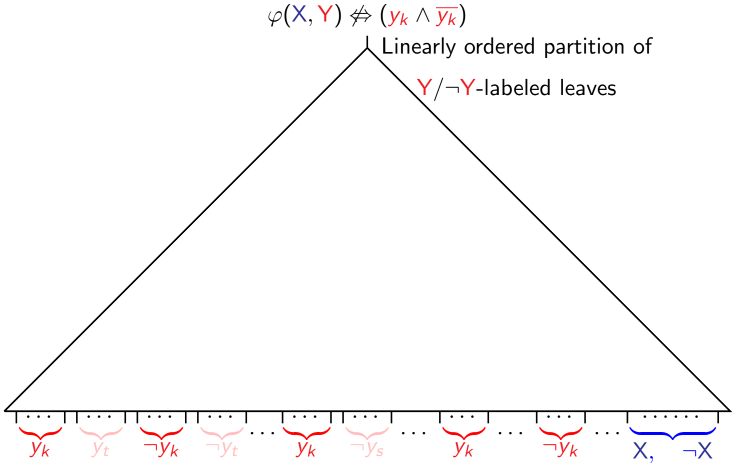
Yes, there exists such a class!

Subset-And-Unrealizable Normal Form (SAUNF)

SAUNF: A Very Special Normal Form

Generalizing forbidden **semantics** of SynNNF

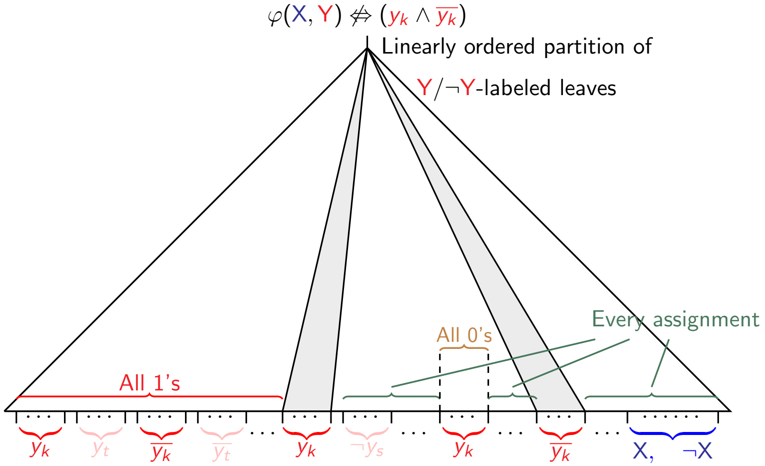
[Shah,Bansal,Akshay,Chakraborty LICS21]



SAUNF: A Very Special Normal Form

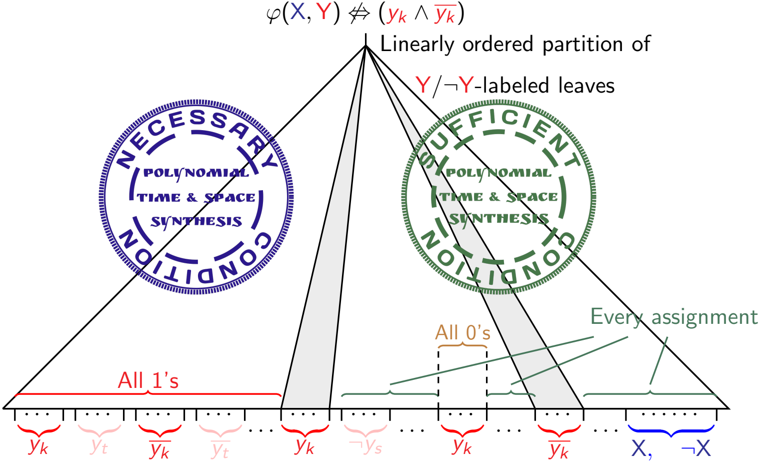
Generalizing forbidden **semantics** of SynNNF

[Shah,Bansal,Akshay,Chakraborty LICS21]



Generalizing forbidden **semantics** of SynNNF

[Shah,Bansal,Akshay,Chakraborty LICS21]



SAUNF vs Existing Popular Normal Forms

Proposition

- Every SynNNF, wDNNF, DNNF circuit is also in SAUNF.
- Every FBDD, ROBDD can be compiled in linear time to SAUNF.

SAUNF vs Existing Popular Normal Forms

Proposition

- Every SynNNF, wDNNF, DNNF circuit is also in SAUNF.
- Every FBDD, ROBDD can be compiled in linear time to SAUNF.

Proposition

SAUNF is strictly more succinct than SynNNF, wDNNF, DNNF, FBDD, ROBDD

SAUNF vs Existing Popular Normal Forms

Proposition

- Every SynNNF, wDNNF, DNNF circuit is also in SAUNF.
- Every FBDD, ROBDD can be compiled in linear time to SAUNF.

Proposition

SAUNF is strictly more succinct than SynNNF, wDNNF, DNNF, FBDD, ROBDD

Proposition

SAUNF is exponentially more succinct than DNNF/dDNNF

SAUNF vs Existing Popular Normal Forms

Proposition

- Every SynNNF, wDNNF, DNNF circuit is also in SAUNF.
- Every FBDD, ROBDD can be compiled in linear time to SAUNF.

Proposition

SAUNF is strictly more succinct than SynNNF, wDNNF, DNNF, FBDD, ROBDD

Proposition

SAUNF is exponentially more succinct than DNNF/dDNNF, which are themselves exponentially more succinct than ROBDDs/FBDD.

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.
  - Otherwise,
    - Not possible in poly-time unless P = NP
    - Not possible in poly-size unless $\Sigma_2^p = \Pi_2^p$

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.
  - Otherwise,
    - Not possible in poly-time unless P = NP
    - Not possible in poly-size unless $\Sigma_2^p = \Pi_2^p$
- Existentially quantifying $y_1, \ldots y_m$ takes linear time.

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.
  - Otherwise,
    - Not possible in poly-time unless $P = NP$
    - Not possible in poly-size unless $\Sigma_2^p = \Pi_2^p$
- Existentially quantifying $y_1, \ldots y_m$ takes linear time.
  - Quantifying subset of Y not possible in linear time in general.

Properties of SAUNF

Operations on SAUNF

Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.
  - Otherwise,
    - Not possible in poly-time unless $P = NP$
    - Not possible in poly-size unless $\Sigma_2^p = \Pi_2^p$
- Existentially quantifying $y_1, \ldots y_m$ takes linear time.
  - Quantifying subset of Y not possible in linear time in general.

Checking if a given specification is in SAUNF

- Is Co-NP complete, given linearly ordered partition of Y-labeled leaves

Properties of SAUNF

Operations on SAUNF

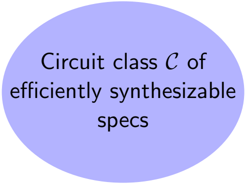
Given $\varphi_1(X, Y)$ and $\varphi_2(X, Y)$ in SAUNF

- Computing $\varphi_1 \vee \varphi_2$ in SAUNF takes constant time
- Computing $\varphi_1 \wedge \varphi_2$ in SAUNF
  - Takes constant time if every pair of Y-labeled leaves of φ_1 and φ_2 are consistent.
  - Otherwise,
    - Not possible in poly-time unless $\mathrm{P} = \mathrm{NP}$
    - Not possible in poly-size unless $\Sigma_2^p = \Pi_2^p$
- Existentially quantifying $y_1, \ldots y_m$ takes linear time.
  - Quantifying subset of Y not possible in linear time in general.

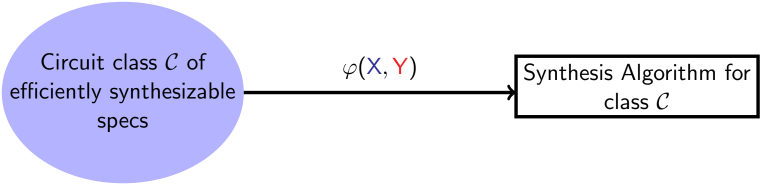
Checking if a given specification is in SAUNF

- Is Co-NP complete, given linearly ordered partition of Y-labeled leaves
- Is Co-NP hard and in Σ_2^P, otherwise

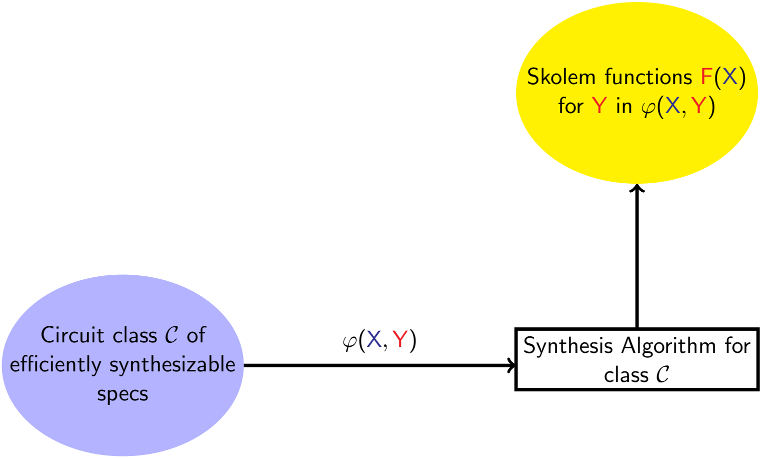
Compiling Other Circuit Classes of Specs to SAUNF



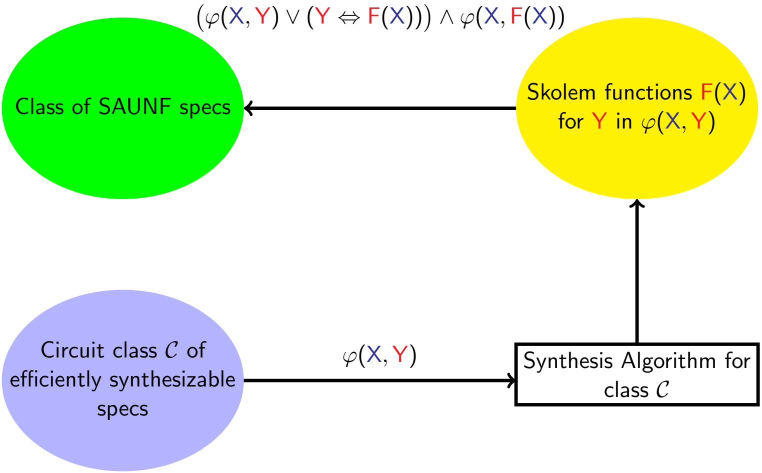
Compiling Other Circuit Classes of Specs to SAUNF



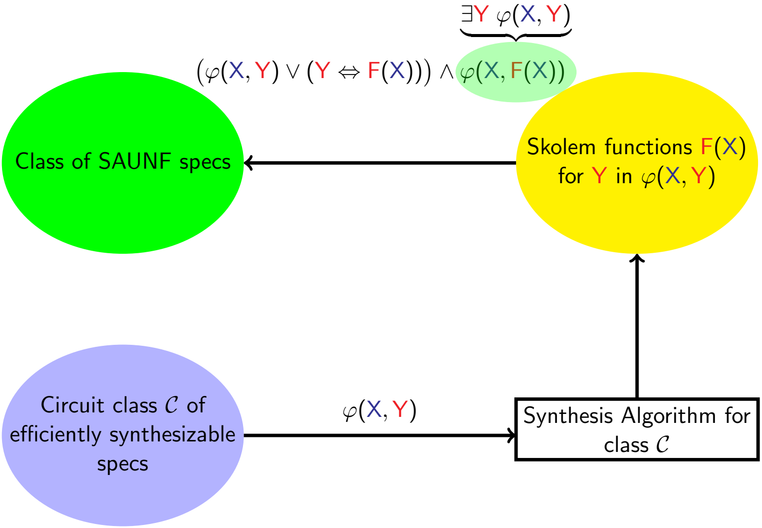
Compiling Other Circuit Classes of Specs to SAUNF



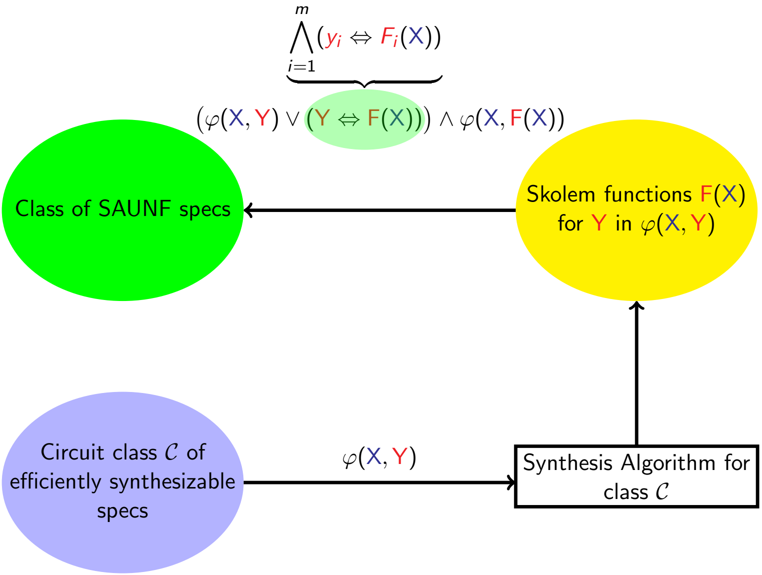
Compiling Other Circuit Classes of Specs to SAUNF



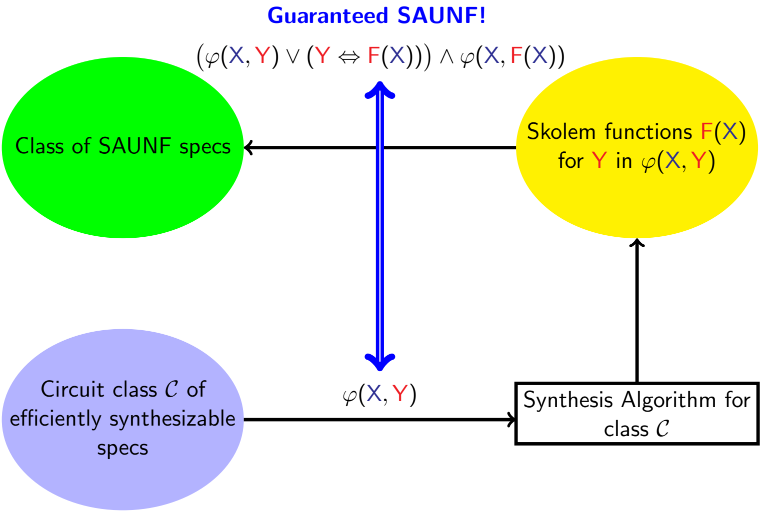
Compiling Other Circuit Classes of Specs to SAUNF



Compiling Other Circuit Classes of Specs to SAUNF



Compiling Other Circuit Classes of Specs to SAUNF



More about compilation to SAUNF

- Easy if class of specs admits efficient synthesis

More about compilation to SAUNF

- Easy if class of specs admits efficient synthesis
- What about other classes of specs?
    - CNF specs: NNF circuits don't always admit efficient synthesis

More about compilation to SAUNF

- Easy if class of specs admits efficient synthesis
- What about other classes of specs?
  - CNF specs: NNF circuits don't always admit efficient synthesis

Compiling CNF to SAUNF [Shah et al LICS21]

- We give an algorithm to compile a CNF formula into SAUNF
- Worst-case exponential-time and space
  - Unavoidable due to hardness results
- Future work: Implementation and comparisons!

An Interesting Application

- n-bit integer outputs Y_1, Y_2; $2n$ bit integer input X

An Interesting Application

- n-bit integer outputs Y_1, Y_2; $2n$ bit integer input X
- Spec $\varphi_{\ell,j}(X, Y_1, Y_2) \equiv \bigwedge_{i=\ell}^{j} \left(X[i] \Leftrightarrow \left(Y_1 \times_{[n]} Y_2\right)[i]\right)$, for $1 \leq \ell \leq j \leq 2n$

An Interesting Application

- n-bit integer outputs Y_1, Y_2; $2n$ bit integer input X
- Spec $\varphi_{\ell,j}(X, Y_1, Y_2) \equiv \bigwedge_{i=\ell}^{j} \left(X[i] \Leftrightarrow \left(Y_1 \times_{[n]} Y_2\right)[i]\right)$, for $1 \leq \ell \leq j \leq 2n$
- $\varphi_{1,2n}(X, Y_1, Y_2) \wedge (Y_1 \neq_{[n]} 1) \wedge (Y_2 \neq_{[n]} 1)$ specifies non-trivial factorization of X

An Interesting Application

- n-bit integer outputs Y_1, Y_2; $2n$ bit integer input X
- Spec $\varphi_{\ell,j}(X, Y_1, Y_2) \equiv \bigwedge_{i=\ell}^{j} \left(X[i] \Leftrightarrow (Y_1 \times_{[n]} Y_2)[i]\right)$, for $1 \leq \ell \leq j \leq 2n$
- $\varphi_{1,2n}(X, Y_1, Y_2) \wedge (Y_1 \neq_{[n]} 1) \wedge (Y_2 \neq_{[n]} 1)$ specifies non-trivial factorization of X

Some initial results [Shah et al LICS21]

- For $1 \leq \ell \leq j \leq 2n$ and $j - \ell < n$, the spec $\varphi_{\ell,j}(X, Y_1, Y_2)$ representable by a poly-sized SAUNF circuit.
- $\varphi_{n,n}(X, Y_1, Y_2)$ has exponential size lower bounds for ROBDDs; sub-exponential representations using DNNF, dDNNF, wDNNF, SynNNF not known.

An Interesting Application

- n-bit integer outputs Y_1, Y_2; $2n$ bit integer input X
- Spec $\varphi_{\ell,j}(X, Y_1, Y_2) \equiv \bigwedge_{i=\ell}^{j} \left(X[i] \Leftrightarrow (Y_1 \times_{[n]} Y_2)[i]\right)$, for $1 \leq \ell \leq j \leq 2n$
- $\varphi_{1,2n}(X, Y_1, Y_2) \wedge (Y_1 \neq_{[n]} 1) \wedge (Y_2 \neq_{[n]} 1)$ specifies non-trivial factorization of X

Some initial results [Shah et al LICS21]

- For $1 \leq \ell \leq j \leq 2n$ and $j - \ell < n$, the spec $\varphi_{\ell,j}(X, Y_1, Y_2)$ representable by a poly-sized SAUNF circuit.
- $\varphi_{n,n}(X, Y_1, Y_2)$ has exponential size lower bounds for ROBDDs; sub-exponential representations using DNNF, dDNNF, wDNNF, SynNNF not known.
- Does not solve factorization (yet!)

Conclusion and Future Work

- A new normal form (SAUNF) that characterizes poly-time/size Boolean Skolem function synthesis.
- Many beautiful and useful properties.
- A compilation algorithm from CNF to SAUNF

Conclusion and Future Work

- A new normal form (SAUNF) that characterizes poly-time/size Boolean Skolem function synthesis.
- Many beautiful and useful properties.
- A compilation algorithm from CNF to SAUNF

Future Work

Conclusion and Future Work

- A new normal form (SAUNF) that characterizes poly-time/size Boolean Skolem function synthesis.
- Many beautiful and useful properties.
- A compilation algorithm from CNF to SAUNF

Future Work

- Implementation of CNF to SAUNF compilation
  - Practical performance studies
  - Fine-tuning of algorithm based on empirical performance

Conclusion and Future Work

- A new normal form (SAUNF) that characterizes poly-time/size Boolean Skolem function synthesis.
- Many beautiful and useful properties.
- A compilation algorithm from CNF to SAUNF

Future Work

- Implementation of CNF to SAUNF compilation
  - Practical performance studies
  - Fine-tuning of algorithm based on empirical performance
- Closing the complexity gap for checking if a specification is in SAUNF

Conclusion and Future Work

- A new normal form (SAUNF) that characterizes poly-time/size Boolean Skolem function synthesis.
- Many beautiful and useful properties.
- A compilation algorithm from CNF to SAUNF

Future Work

- Implementation of CNF to SAUNF compilation
  - Practical performance studies
  - Fine-tuning of algorithm based on empirical performance
- Closing the complexity gap for checking if a specification is in SAUNF
- Going beyond the Boolean case!

Thank you!