Knowledge Compilation for Boolean Functional Synthesis

S. Akshay, Jatin Arora, Supratik Chakraborty, S. Krishna,
Divya Raghunathan, Shetal Shah

Indian Institute of Technology Bombay

Boolean Functional Synthesis

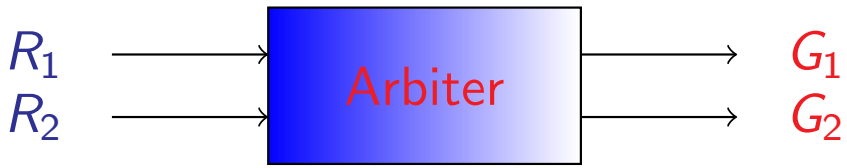
- Boolean functions: fundamental building blocks in computing.

Boolean Functional Synthesis

- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs

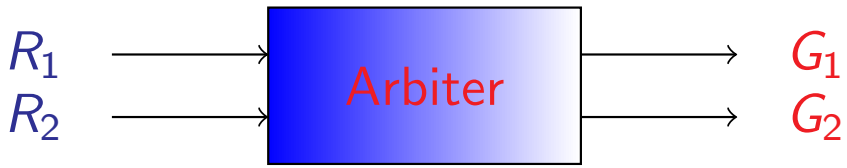
Boolean Functional Synthesis

- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs
  - E.g. Arbiter



Boolean Functional Synthesis

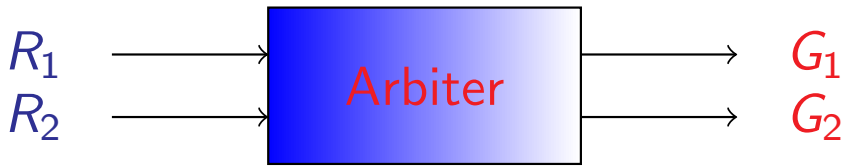
- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs
  - E.g. Arbiter



  - $(R_1 \vee R_2) \rightarrow (G_1 \vee G_2) \wedge \neg(G_1 \wedge G_2) \wedge (G_1 \rightarrow R_1) \wedge (G_2 \rightarrow R_2)$

Boolean Functional Synthesis

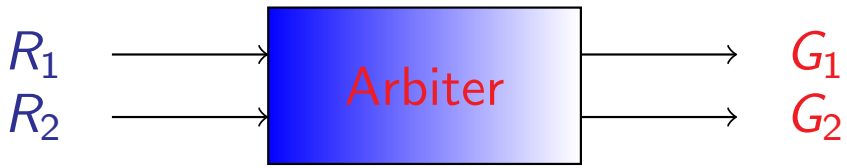
- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs
  - E.g. Arbiter



  - $(R_1 \vee R_2) \rightarrow (G_1 \vee G_2) \wedge \neg(G_1 \wedge G_2) \wedge (G_1 \rightarrow R_1) \wedge (G_2 \rightarrow R_2)$
- But we need them in *functional form*
  - Outputs as functions of inputs

Boolean Functional Synthesis

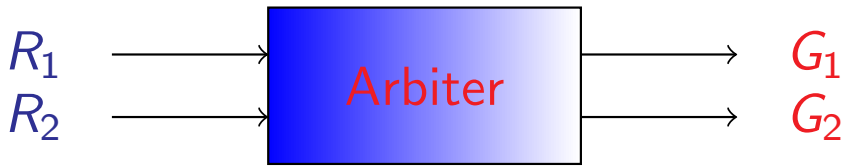
- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs
  - E.g. Arbiter



  - $(R_1 \vee R_2) \rightarrow (G_1 \vee G_2) \wedge \neg(G_1 \wedge G_2) \wedge (G_1 \rightarrow R_1) \wedge (G_2 \rightarrow R_2)$
- But we need them in *functional form*
  - Outputs as functions of inputs
  - $G_1 = Func1(R_1, R_2)$, $G_2 = Func2(R_1, R_2)$,

Boolean Functional Synthesis

- Boolean functions: fundamental building blocks in computing.
- Often easy to specify *relationally*;
  - Relation between inputs and outputs
  - E.g. Arbiter



  - $(R_1 \vee R_2) \rightarrow (G_1 \vee G_2) \wedge \neg(G_1 \wedge G_2) \wedge (G_1 \rightarrow R_1) \wedge (G_2 \rightarrow R_2)$
- But we need them in *functional form*
  - Outputs as functions of inputs
  - $G_1 = Func1(R_1, R_2)$, $G_2 = Func2(R_1, R_2)$,

Boolean Functional Synthesis (BFnS)

Synthesizing Boolean functions from a relational specification.

Boolean Functional Synthesis BFnS

Formal definition

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

- x_i *input* variables
- y_j *output* variables,

Boolean Functional Synthesis BFnS

Formal definition

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

- x_i *input* variables
- y_j *output* variables,

Synthesize Boolean functions $F_j(x_1 \ldots x_n)$ for each y_j s.t.

$\forall x_1 \ldots x_n \, (\exists y_1 \ldots y_m \; \varphi(x_1 \ldots x_n, y_1 \ldots y_m) \;\Leftrightarrow\; \varphi(x_1, \ldots x_n, F_1, \ldots F_m))$

Boolean Functional Synthesis BFnS

Formal definition

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

- x_i *input* variables
- y_j *output* variables,

Synthesize Boolean functions $F_j(x_1 \ldots x_n)$ for each y_j s.t.

$\forall x_1 \ldots x_n \, (\exists y_1 \ldots y_m \; \varphi(x_1 \ldots x_n, y_1 \ldots y_m) \;\Leftrightarrow\; \varphi(x_1, \ldots x_n, F_1, \ldots F_m))$

F_j is also called a *Skolem function* for y_j in φ.

Boolean Functional Synthesis BFnS

Formal definition

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

- x_i *input* variables
- y_j *output* variables,

Synthesize Boolean functions $F_j(x_1 \ldots x_n)$ for each y_j s.t.

$\forall x_1 \ldots x_n \, (\exists y_1 \ldots y_m \; \varphi(x_1 \ldots x_n, y_1 \ldots y_m) \;\Leftrightarrow\; \varphi(x_1, \ldots x_n, F_1, \ldots F_m))$

F_j is also called a *Skolem function* for y_j in φ.

Interesting even if for some $x_1, \ldots x_n$, there isn't $y_1, \ldots y_m$ s.t. $\varphi(x_1 \ldots x_n, y_1 \ldots y_m)$ holds.

Applications of Boolean Functional Synthesis

1. Factorization: Interesting but hard for synthesis!
2. Disjunctive decomposition of transition relations [Trivedi et al'02]
3. Quantifier elimination, of course!
   - $\exists Y\ \varphi(X, Y) \equiv \varphi(X, F(X))$
4. Certifying QBF-SAT solvers [Bendetti'05, ...]
5. (Reactive) circuit/strategy synthesis [Alur et al'05, ...]
6. Program synthesis
   - Combinatorial sketching [Solar-Lezama et al'06, Srivastava et al'13, ...]
   - Complete functional synthesis [Kuncak et al'10, ...]
7. Repair/partial synthesis of circuits [Fujita et al'13 ...]

Existing Approaches

1. Closely related to Boolean unification
   1. Given $\varphi(\mathbf{X}, y_1, .., y_n) = 0$, find most general unifiers of $y_1, .., y_m$ in terms of $\mathbf{X}$.
      - Early work by Boole (1847) and Lowenheim (1908)

Existing Approaches

1. Closely related to Boolean unification
   1. Given $\varphi(\mathbf{X}, y_1, .., y_n) = 0$, find most general unifiers of $y_1, .., y_m$ in terms of $\mathbf{X}$.
      - Early work by Boole (1847) and Lowenheim (1908)
2. Extract Skolem functions from proof of validity of $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$
   - Bendetti'05, Jussilla et al'07, Jiang et al'11, Heule et al'14, ...
   - Efficient if a short proof of validity is found
   - Doesn't work if $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ is not valid.

Existing Approaches

1. Closely related to Boolean unification
   1. Given $\varphi(\mathbf{X}, y_1, .., y_n) = 0$, find most general unifiers of $y_1, .., y_m$ in terms of $\mathbf{X}$.
      - Early work by Boole (1847) and Lowenheim (1908)
2. Extract Skolem functions from proof of validity of $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$
   - Bendetti'05, Jussilla et al'07, Jiang et al'11, Heule et al'14, ...
   - Efficient if a short proof of validity is found
   - Doesn't work if $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ is not valid.
3. Generate Skolem functions matching given templates
   - Srivastava et al'13, ...
   - Effective when small set of candidate Skolem functions known
   - Not always the case

Existing Approaches

4. Function composition based approaches
   - Jiang'09, Trivedi et al'03
   - Iterated compositions cause formula blow up

Existing Approaches

4. Function composition based approaches
   - Jiang'09, Trivedi et al'03
   - Iterated compositions cause formula blow up
5. Boolean functional synthesis using BDDs
   - Kukula et al'00, Kuncak et al'10, Fried et al'16, Tabajara et al'17
   - Scales for a class of benchmarks with pre-determined var orders
   - Sans good var orders, performance degrades considerably

Existing Approaches

4. Function composition based approaches
   - Jiang'09, Trivedi et al'03
   - Iterated compositions cause formula blow up
5. Boolean functional synthesis using BDDs
   - Kukula et al'00, Kuncak et al'10, Fried et al'16, Tabajara et al'17
   - Scales for a class of benchmarks with pre-determined var orders
   - Sans good var orders, performance degrades considerably
6. Incremental determinization
   - Rabe et al'17
   - Works well for a large class of PCNF inputs
   - Doesn't work if $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ is not valid

Existing Approaches

4. Function composition based approaches
   - Jiang'09, Trivedi et al'03
   - Iterated compositions cause formula blow up
5. Boolean functional synthesis using BDDs
   - Kukula et al'00, Kuncak et al'10, Fried et al'16, Tabajara et al'17
   - Scales for a class of benchmarks with pre-determined var orders
   - Sans good var orders, performance degrades considerably
6. Incremental determinization
   - Rabe et al'17
   - Works well for a large class of PCNF inputs
   - Doesn't work if $\forall \mathbf{X} \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ is not valid
7. Quantifier instantiation techniques in SMT solvers
   - Barrett et al'15, Bierre et al'17
   - Works even for bit-vector and other theories
   - May not scale well for large specs with large # quantified vars
8. Input-output separation: Tabajara et al'18
9. Initial approximation iterative refinement Akshay et al'16

How Hard (or Easy) Is It?

- BFnS is *NP*-hard
  - Consider a co-NP^{NP} (or Π_2^P)-complete problem:
    - Is $\forall x_1, \ldots x_n \exists y_1, \ldots y_m \varphi(x_1, .., x_n, y_1, .., y_m)$ true?

How Hard (or Easy) Is It?

- BFnS is *NP*-hard
  - Consider a co-NP$^{\text{NP}}$ (or Π_2^P)-complete problem:
    - Is $\forall x_1, \ldots x_n \exists y_1, \ldots y_m \varphi(x_1, .., x_n, y_1, .., y_m)$ true?
  - Synthesize Skolem function $F_1(x_1 \ldots x_n), \ldots F_m(x_1 \ldots x_n)$, substitute for $y_1, \ldots y_m$ and invoke a co-NP oracle.

How Hard (or Easy) Is It?

- BFnS is *NP*-hard
  - Consider a co-NP$^{\text{NP}}$ (or Π_2^P)-complete problem:
    - Is $\forall x_1, \ldots x_n \exists y_1, \ldots y_m \varphi(x_1, .., x_n, y_1, .., y_m)$ true?
  - Synthesize Skolem function $F_1(x_1 \ldots x_n), \ldots F_m(x_1 \ldots x_n)$, substitute for $y_1, \ldots y_m$ and invoke a co-NP oracle.
  - co-NP$^{\text{NP}}$ included in co-NP$^{\text{BFnS}}$

How Hard (or Easy) Is It?

- BFnS is *NP*-hard
  - Consider a co-NP$^{\text{NP}}$ (or Π_2^P)-complete problem:
    - Is $\forall x_1, \ldots x_n \exists y_1, \ldots y_m \varphi(x_1, .., x_n, y_1, .., y_m)$ true?
  - Synthesize Skolem function $F_1(x_1 \ldots x_n), \ldots F_m(x_1 \ldots x_n)$, substitute for $y_1, \ldots y_m$ and invoke a co-NP oracle.
  - co-NP$^{\text{NP}}$ included in co-NP$^{\text{BFnS}}$
- What about size of Skolem functions?
  - Does there always exist compact Skolem functions, although synthesizing may take exponential time?

How Hard (or Easy) Is It?

- BFnS is *NP*-hard
  - Consider a co-NP$^{\text{NP}}$ (or Π_2^P)-complete problem:
    - Is $\forall x_1, \ldots x_n \exists y_1, \ldots y_m \varphi(x_1, .., x_n, y_1, .., y_m)$ true?
  - Synthesize Skolem function $F_1(x_1 \ldots x_n), \ldots F_m(x_1 \ldots x_n)$, substitute for $y_1, \ldots y_m$ and invoke a co-NP oracle.
  - co-NP$^{\text{NP}}$ included in co-NP$^{\text{BFnS}}$
- What about size of Skolem functions?
  - Does there always exist compact Skolem functions, although synthesizing may take exponential time?
- Lower bound results in circuit-size refer to monotone circuits [Razbarov 1985; Alon and Boppana 1987]
  - Skolem functions need not be monotone
  - Different argument for lower bounds on Skolem circuits

Some Good and Bad News

Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.

Some Good and Bad News

Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.
- Unless non-uniform exponential-time hypothesis fails, ... sizes must be exponential in $|F|$.

Some Good and Bad News

Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.
- Unless non-uniform exponential-time hypothesis fails, ... sizes must be exponential in $|F|$.

Efficient algorithms for Boolean functional synthesis unlikely

Some Good and Bad News

Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.
- Unless non-uniform exponential-time hypothesis fails, ... sizes must be exponential in $|F|$.

Efficient algorithms for Boolean functional synthesis unlikely

Good news: [CAV2018]

- If F is in weak decomposable negation normal form (wDNNF), synthesis solvable in polynomial (in $|F|$) time and space.

Some Good and Bad News

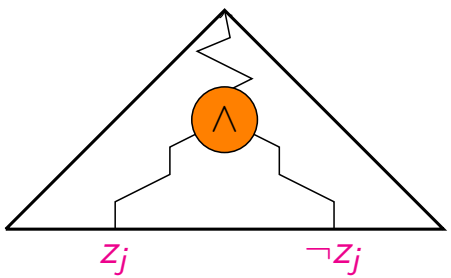
Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.
- Unless non-uniform exponential-time hypothesis fails, ... sizes must be exponential in $|F|$.

Efficient algorithms for Boolean functional synthesis unlikely

Good news: [CAV2018]

- If F is in weak decomposable negation normal form (wDNNF), synthesis solvable in polynomial (in $|F|$) time and space.



wDNNF

Disallowed paths in NNF of φ

Some Good and Bad News

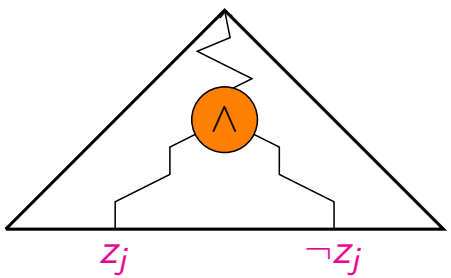
Bad news: [CAV2018]

- Unless $\Pi_2^P = \Sigma_2^P$, there exist specs F for which Skolem function sizes must be super-polynomial in $|F|$.
- Unless non-uniform exponential-time hypothesis fails, ... sizes must be exponential in $|F|$.

Efficient algorithms for Boolean functional synthesis unlikely

Good news: [CAV2018]

- If F is in weak decomposable negation normal form (wDNNF), synthesis solvable in polynomial (in $|F|$) time and space.



wDNNF

Disallowed paths in NNF of φ

Experiments: Efficient synthesis even when F not in wDNNF

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

- **YES: Synthesis Negation Normal Form (SynNNF)**
- Subsumes and exponentially more succinct than BDD/DNNF/wDNNF/...

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

- **YES: Synthesis Negation Normal Form (SynNNF)**
- Subsumes and exponentially more succinct than BDD/DNNF/wDNNF/...

Can we synthesize F from a "simplified" specification?

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

- **YES: Synthesis Negation Normal Form (SynNNF)**
- Subsumes and exponentially more succinct than BDD/DNNF/wDNNF/...

Can we synthesize F from a "simplified" specification?

- **YES:** Folklore wisdom
- **Formalized as refinement w.r.t. synthesis**

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

- **YES: Synthesis Negation Normal Form (SynNNF)**
- Subsumes and exponentially more succinct than BDD/DNNF/wDNNF/...

Can we synthesize F from a "simplified" specification?

- **YES:** Folklore wisdom
- **Formalized as refinement w.r.t. synthesis**

Can we algorithmically compile F to a refined SynNNF spec $\widetilde{F}$?

Current Talk

Is there a weaker (than wDNNF) representation form of F that guarantees poly-time (in $|F|$) synthesis?

- **YES: Synthesis Negation Normal Form (SynNNF)**
- Subsumes and exponentially more succinct than BDD/DNNF/wDNNF/...

Can we synthesize F from a "simplified" specification?

- **YES:** Folklore wisdom
- **Formalized as refinement w.r.t. synthesis**

Can we algorithmically compile F to a refined SynNNF spec $\widetilde{F}$?

- **YES:** Super-polynomial time in worst-case
- **Practical performance promising!**

Classical Knowledge Compilation

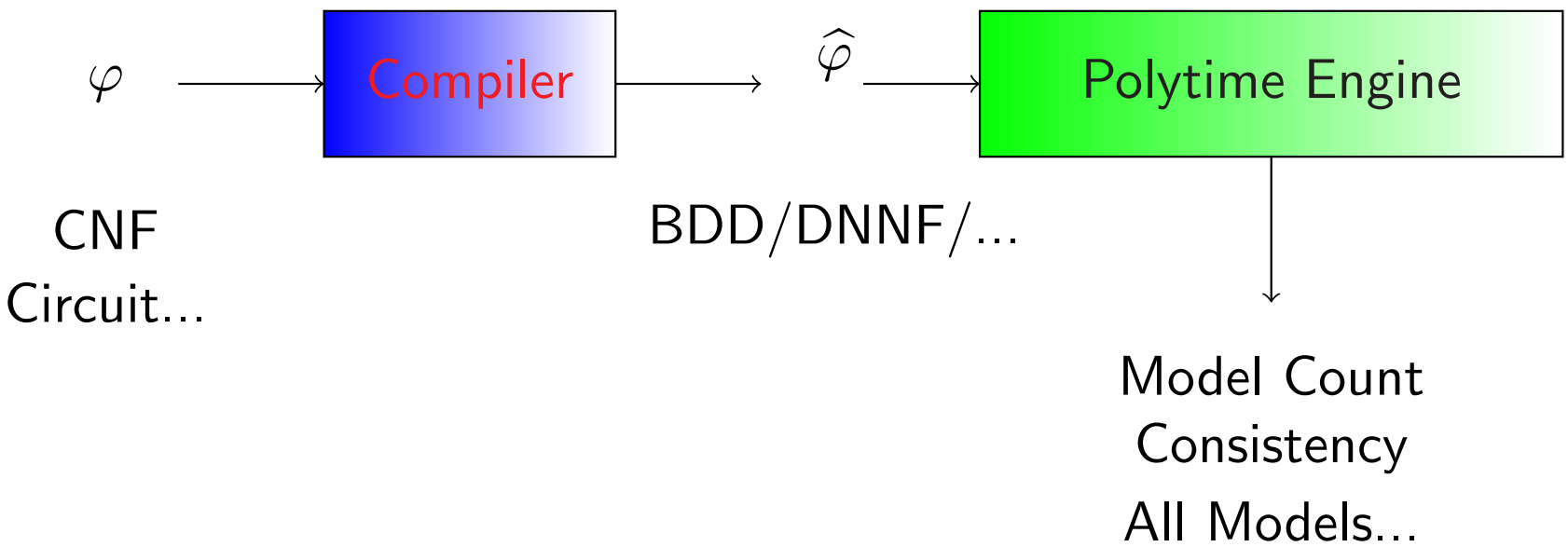
Wikipedia

... a family of approaches for addressing the intractability of a number of artificial intelligence problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.

Classical Knowledge Compilation

Wikipedia

... a family of approaches for addressing the intractability of a number of artificial intelligence problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.



Knowledge Compilation for Synthesis

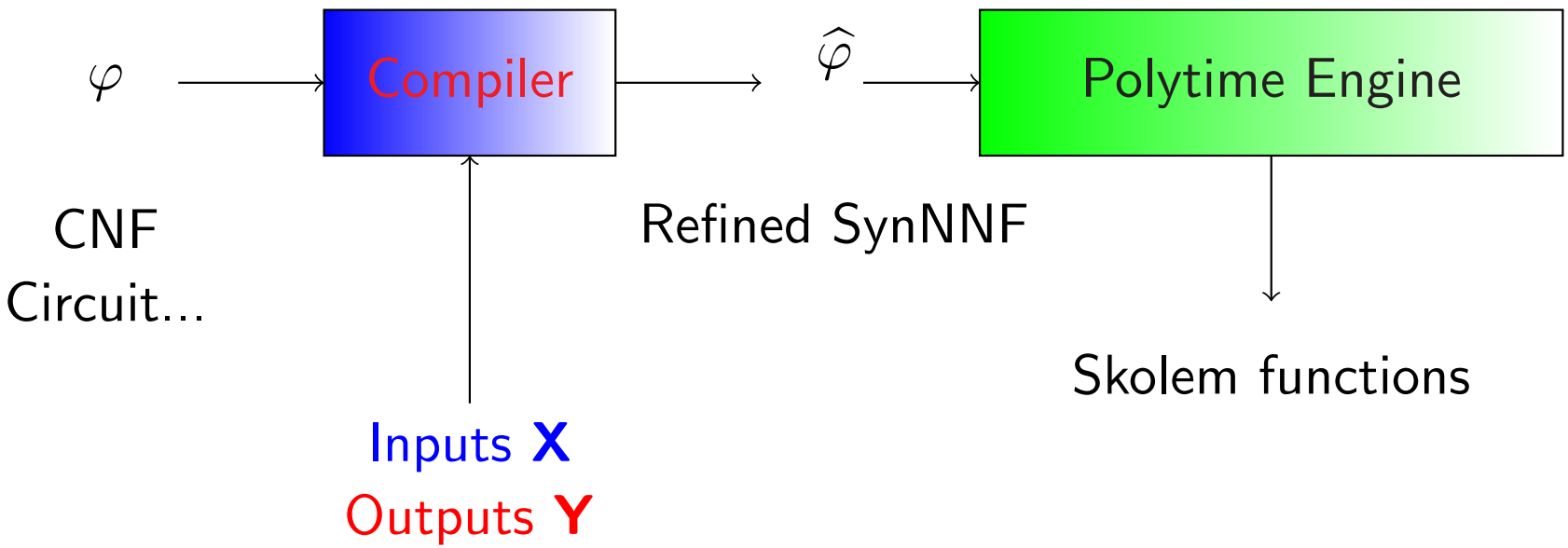
Our Definition

... a family of approaches for addressing the intractability of synthesis problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.

Knowledge Compilation for Synthesis

Our Definition

... a family of approaches for addressing the intractability of synthesis problems. A propositional model is compiled in an off-line phase in order to support some queries in polytime.



SynNNF: A Quick Introduction

- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ as NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves

SynNNF: A Quick Introduction

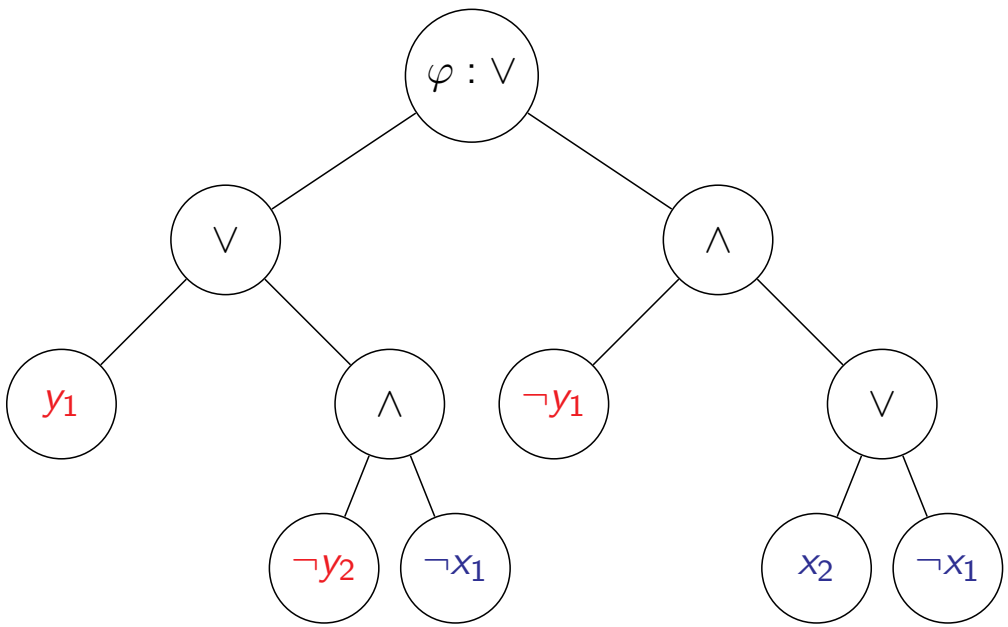
- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ as NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves

Ex: $\varphi(x_1, x_2, y_1, y_2) \equiv (y_1 \vee (\neg y_2 \wedge \neg x_1)) \vee (\neg y_1 \wedge (x_2 \vee \neg x_1))$

SynNNF: A Quick Introduction

- Represent $\varphi(x_1, .., x_n, y_1, .., y_m)$ as NNF DAG
  - Boolean circuit, $\wedge$ and $\vee$ at internal nodes, $\neg$ only at leaves

Ex: $\varphi(x_1, x_2, y_1, y_2) \equiv (y_1 \vee (\neg y_2 \wedge \neg x_1)) \vee (\neg y_1 \wedge (x_2 \vee \neg x_1))$



Positive form of specification

- Define $\widehat{\varphi}(x_1, \ldots, x_n, y_1, \ldots, y_m, \overline{y_1}, \ldots \overline{y_m})$
  - In NNF DAG of φ, replace every $\neg y_i$ leaf by fresh variable $\overline{y_i}$

Positive form of specification

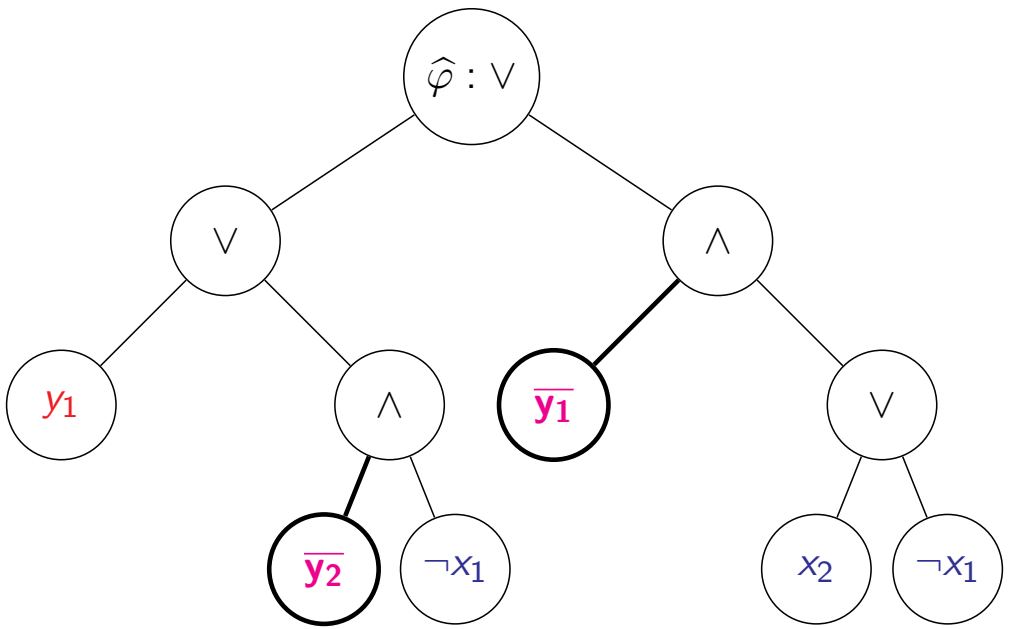
- Define $\widehat{\varphi}(x_1, \ldots, x_n, y_1, \ldots, y_m, \overline{y_1}, \ldots \overline{y_m})$
  - In NNF DAG of φ, replace every $\neg y_i$ leaf by fresh variable $\overline{y_i}$

Ex: $\varphi(x_1, x_2, y_1, y_2) \equiv (y_1 \vee (\neg y_2 \wedge \neg x_1)) \vee (\neg y_1 \wedge (x_2 \vee \neg x_1))$
$\widehat{\varphi}(x_1, x_2, y_1, y_2, \overline{y_1}, \overline{y_2}) \equiv (y_1 \vee (\overline{y_2} \wedge \neg x_1)) \vee (\overline{y_1} \wedge x_2 \vee \neg x_1)$

Positive form of specification

- Define $\widehat{\varphi}(x_1, \ldots, x_n, y_1, \ldots, y_m, \overline{y_1}, \ldots \overline{y_m})$
  - In NNF DAG of φ, replace every $\neg y_i$ leaf by fresh variable $\overline{y_i}$

Ex: $\varphi(x_1, x_2, y_1, y_2) \equiv (y_1 \vee (\neg y_2 \wedge \neg x_1)) \vee (\neg y_1 \wedge (x_2 \vee \neg x_1))$
$\widehat{\varphi}(x_1, x_2, y_1, y_2, \overline{y_1}, \overline{y_2}) \equiv (y_1 \vee (\overline{y_2} \wedge \neg x_1)) \vee (\overline{y_1} \wedge x_2 \vee \neg x_1)$

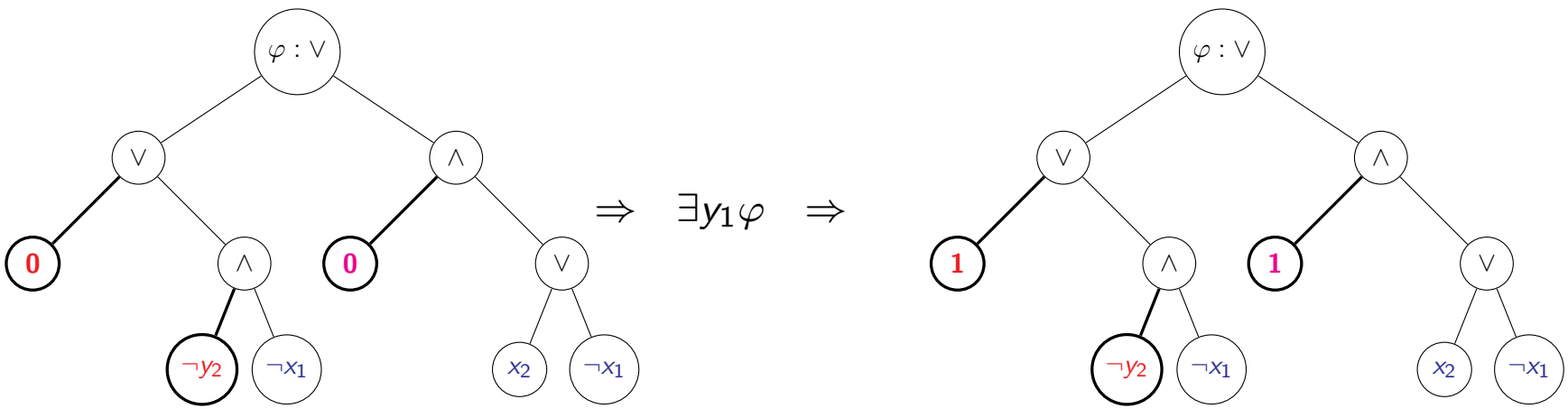


Simple properties of $\widehat{\varphi}$

- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{0..0}^{i}, y_{i+1} \ldots y_m, \overbrace{0..0}^{i}, \neg y_{i+1} \ldots \neg y_m) \Rightarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$
- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{1..1}^{i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i}, \neg y_{i+1} \ldots \neg y_m) \Leftarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$

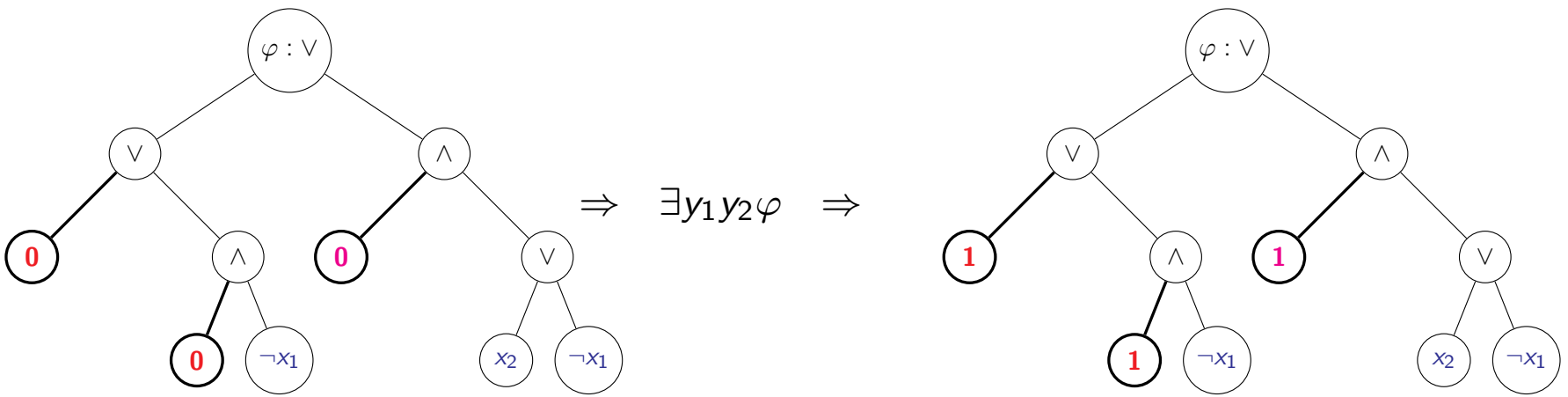
Simple properties of $\widehat{\varphi}$

- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{0..0}^{i}, y_{i+1} \ldots y_m, \overbrace{0..0}^{i}, \neg y_{i+1} \ldots \neg y_m) \Rightarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$
- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{1..1}^{i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i}, \neg y_{i+1} \ldots \neg y_m) \Leftarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$



Simple properties of $\widehat{\varphi}$

- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{0..0}^{i}, y_{i+1} \ldots y_m, \overbrace{0..0}^{i}, \neg y_{i+1} \ldots \neg y_m) \Rightarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$
- $\widehat{\varphi}(x_1 \ldots x_n, \overbrace{1..1}^{i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i}, \neg y_{i+1} \ldots \neg y_m) \Leftarrow \exists y_1 \ldots y_i \ \varphi(\ldots)$



SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

Does $\exists x_1, \ldots x_n \exists y_{i+1} \ldots y_m \quad (\ [\widehat{\varphi}]_i \ \Leftrightarrow \ \mathbf{y_i} \wedge \overline{\mathbf{y_i}} \ )$ hold?

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

Does $\exists x_1, \ldots x_n \exists y_{i+1} \ldots y_m \quad (\ [\widehat{\varphi}]_i \ \Leftrightarrow \ \mathbf{y_i} \wedge \overline{\mathbf{y_i}} \ )$ hold?

Can $[\widehat{\varphi}]_i$ be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$?

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

Does $\exists x_1, \ldots x_n \exists y_{i+1} \ldots y_m \quad (\ [\widehat{\varphi}]_i \ \Leftrightarrow \ \mathbf{y_i} \wedge \overline{\mathbf{y_i}} \ )$ hold?

Can $[\widehat{\varphi}]_i$ be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$?

If $[\widehat{\varphi}]_i$ cannot be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

Does $\exists x_1, \ldots x_n \exists y_{i+1} \ldots y_m \quad (\ [\widehat{\varphi}]_i \ \Leftrightarrow \ \mathbf{y_i} \wedge \overline{\mathbf{y_i}} \ )$ hold?

Can $[\widehat{\varphi}]_i$ be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$?

If $[\widehat{\varphi}]_i$ cannot be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$ for all $i \in \{1, \ldots n\}$,

SynNNF: A Quick Introduction

Given

- Positive form of spec $\widehat{\varphi}(x_1, \ldots x_n, y_1, \ldots y_m, \overline{y_1}, \ldots \overline{y_m})$
- Sequence (linear order) of outputs $(y_1, \ldots y_m)$

Let $[\widehat{\varphi}]_i$ be

$$\widehat{\varphi}(x_1, \ldots x_n, \overbrace{1..1}^{i-1}, \mathbf{y_i}, y_{i+1} \ldots y_m, \overbrace{1..1}^{i-1}, \overline{\mathbf{y_i}}, \neg y_{i+1} \ldots \neg y_m).$$

Does $\exists x_1, \ldots x_n \exists y_{i+1} \ldots y_m \quad (\ [\widehat{\varphi}]_i \ \Leftrightarrow \ \mathbf{y_i} \wedge \overline{\mathbf{y_i}} \ )$ hold?

Can $[\widehat{\varphi}]_i$ be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$?

If $[\widehat{\varphi}]_i$ cannot be made to behave like $\mathbf{y_i} \wedge \overline{\mathbf{y_i}}$ for all $i \in \{1, \ldots n\}$,

Representation of φ is in **SynNNF**

SynNNF: Some observations

- Every propositional formula representable in SynNNF for every ordering of outputs
  - DNF always SynNNF for any output order

SynNNF: Some observations

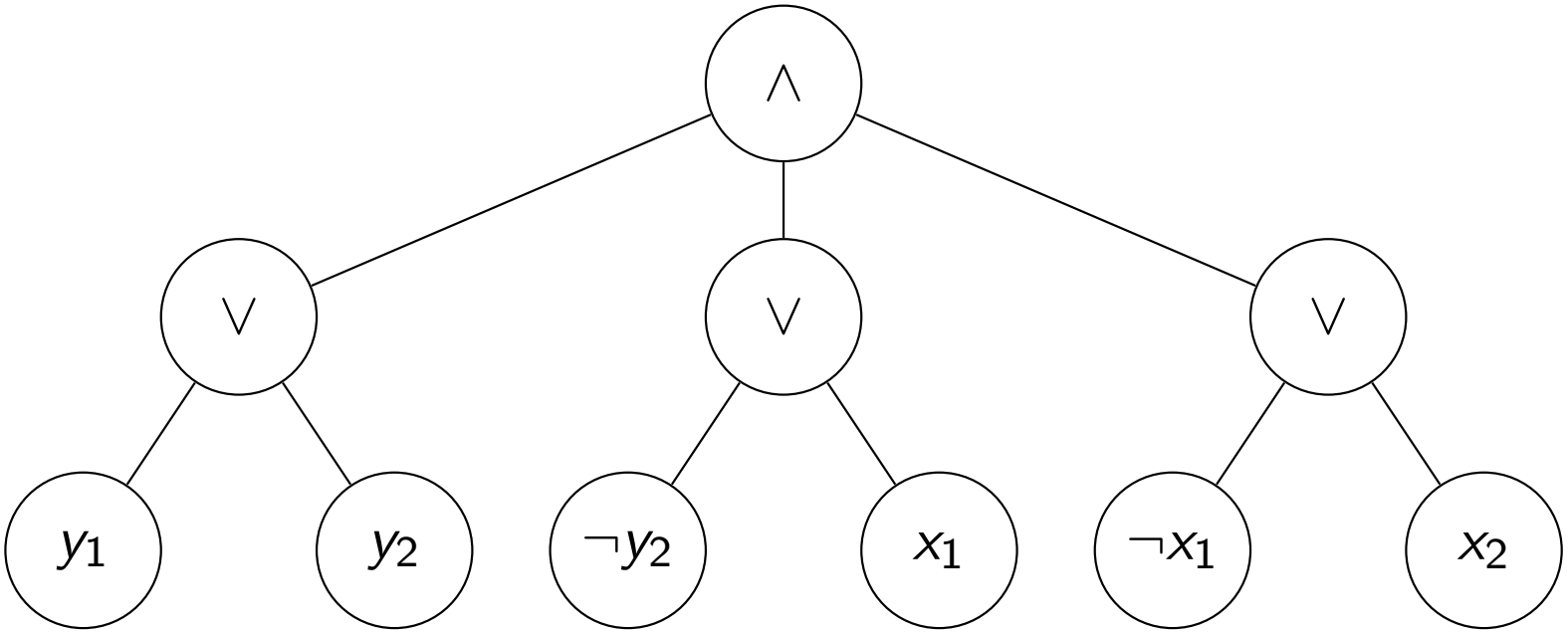
- Every propositional formula representable in SynNNF for every ordering of outputs
  - DNF always SynNNF for any output order
- A formula may have multiple SynNNF representations
  - DNF, BDD, DNNF ...

SynNNF: Some observations

- Every propositional formula representable in SynNNF for every ordering of outputs
  - DNF always SynNNF for any output order
- A formula may have multiple SynNNF representations
  - DNF, BDD, DNNF ...
- A given representation may be SynNNF for order1 of outputs and not in SynNNF for order2 of outputs.
  - Example coming up ...

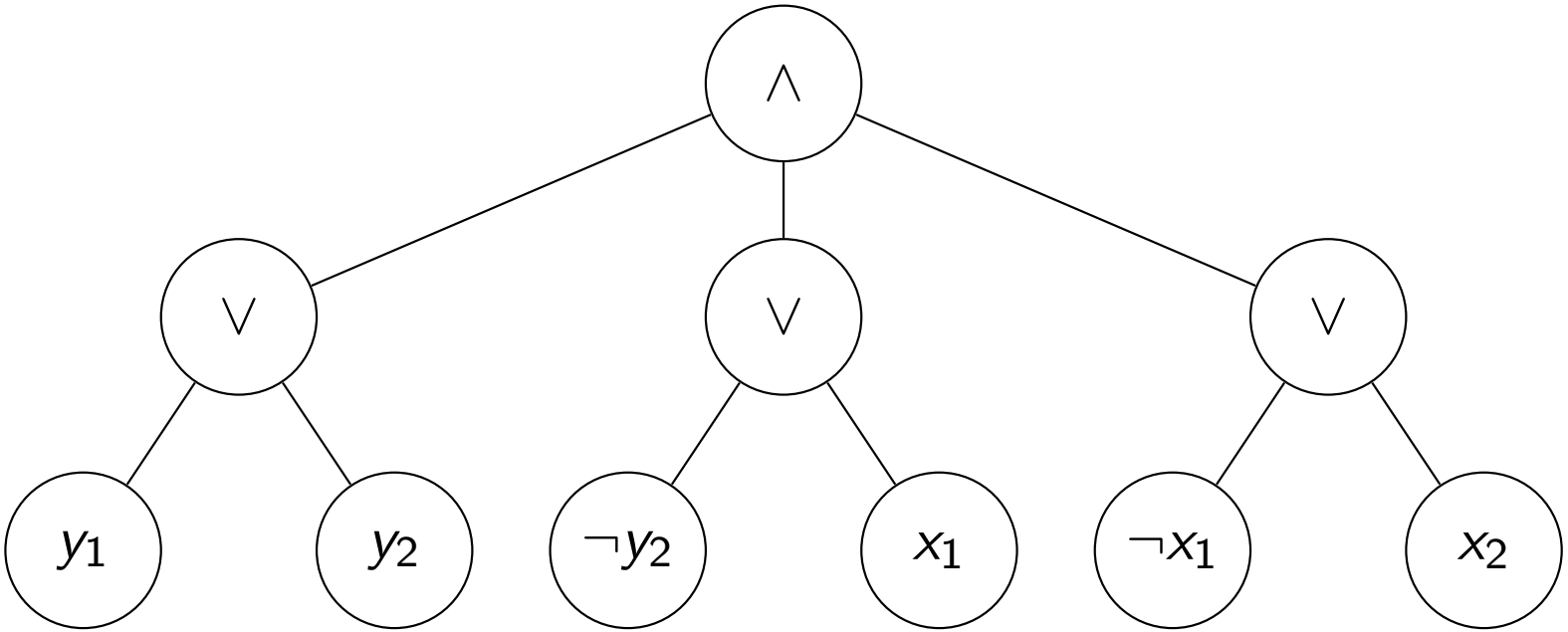
SynNNF: An Example

$$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$$



SynNNF: An Example

$$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$$

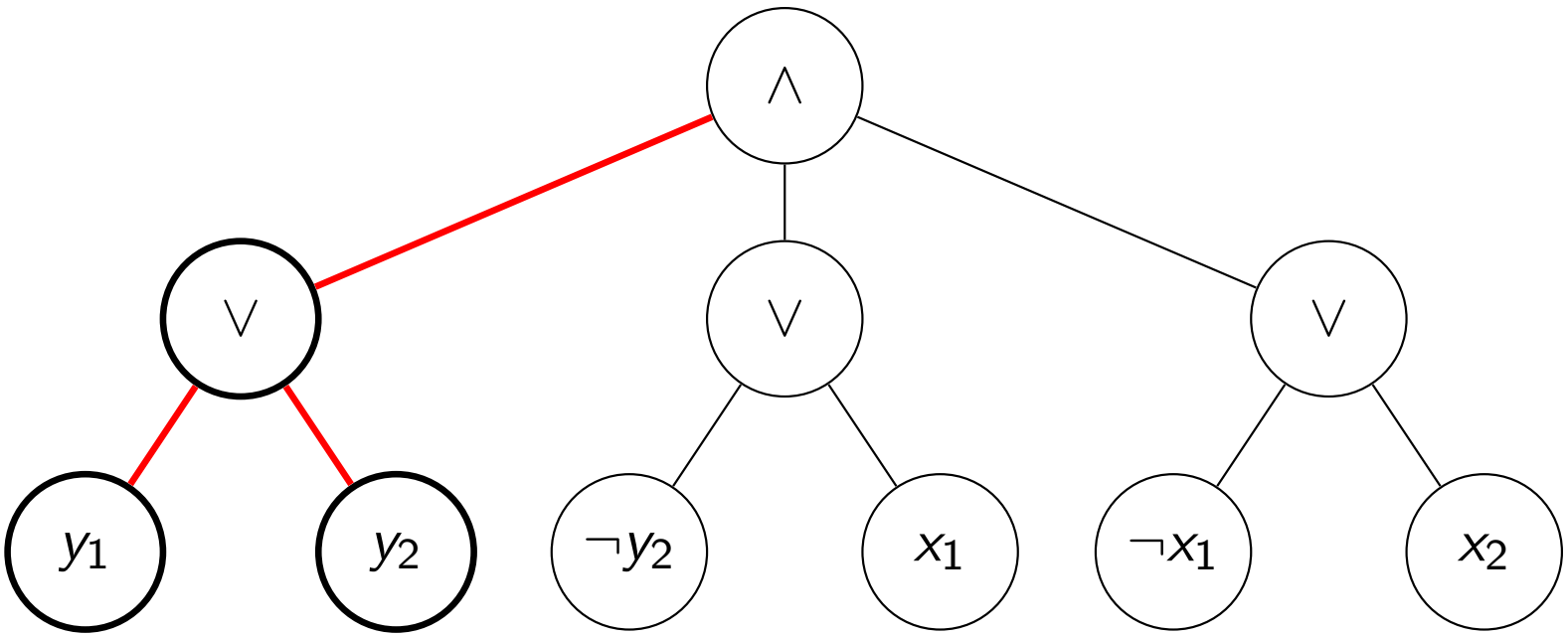


Representation of φ **not** in DNNF/wDNNF

SynNNF: An Example

$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$

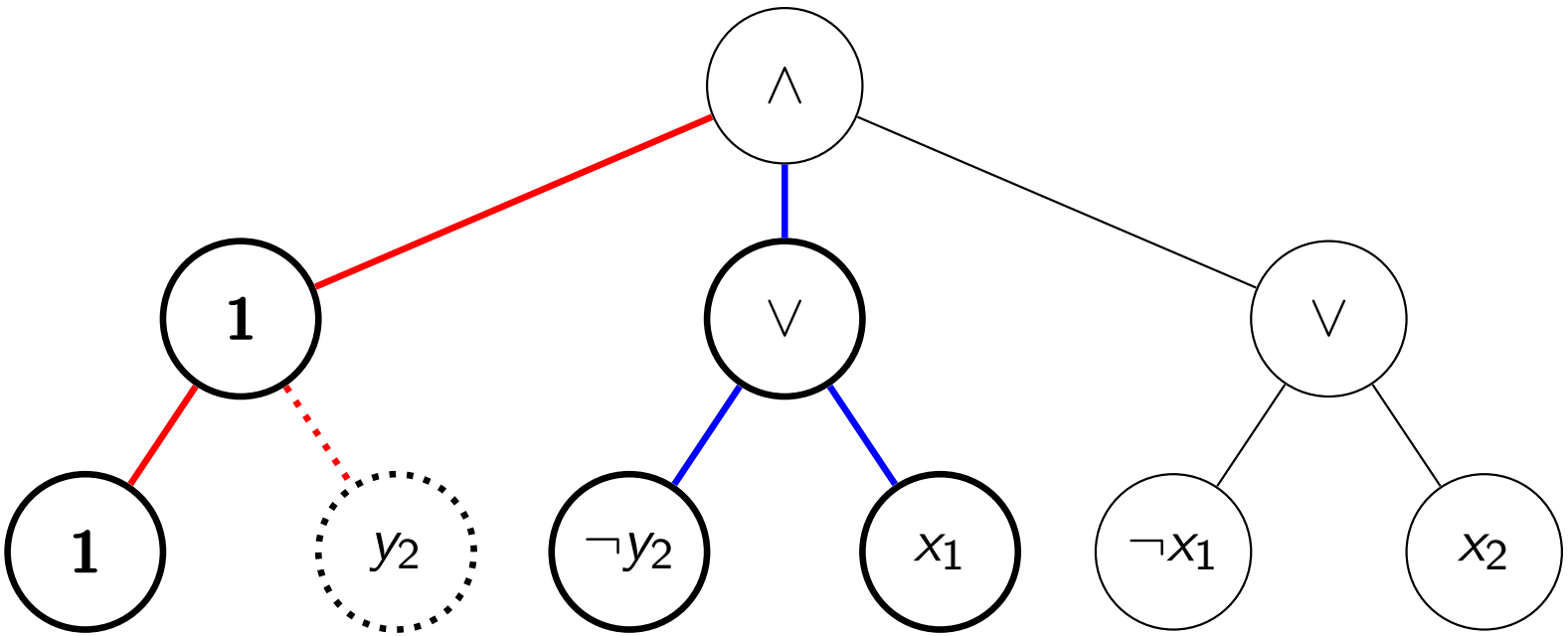
Output sequence: $(\underline{y_1}, y_2)$



SynNNF: An Example

$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$

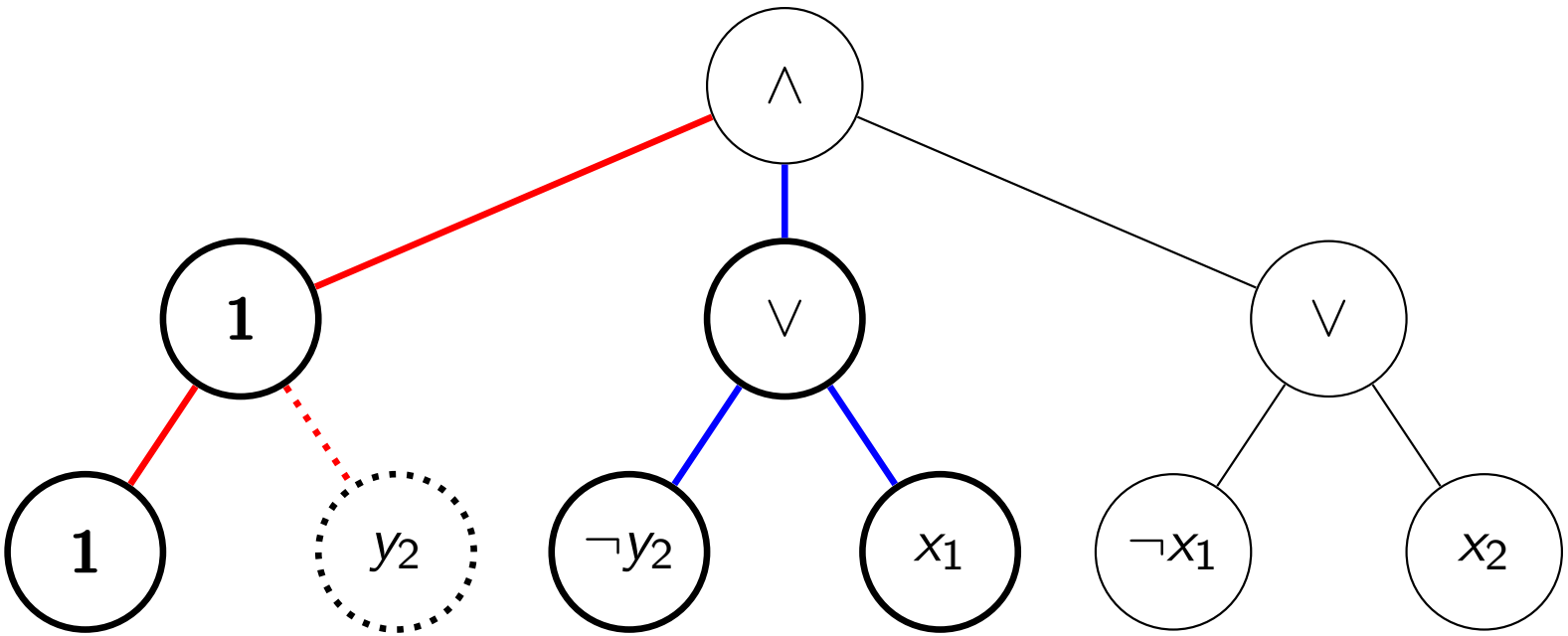
Output sequence: $(y_1, \underline{y_2})$



SynNNF: An Example

$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$

Output sequence: $(y_1, \underline{y_2})$

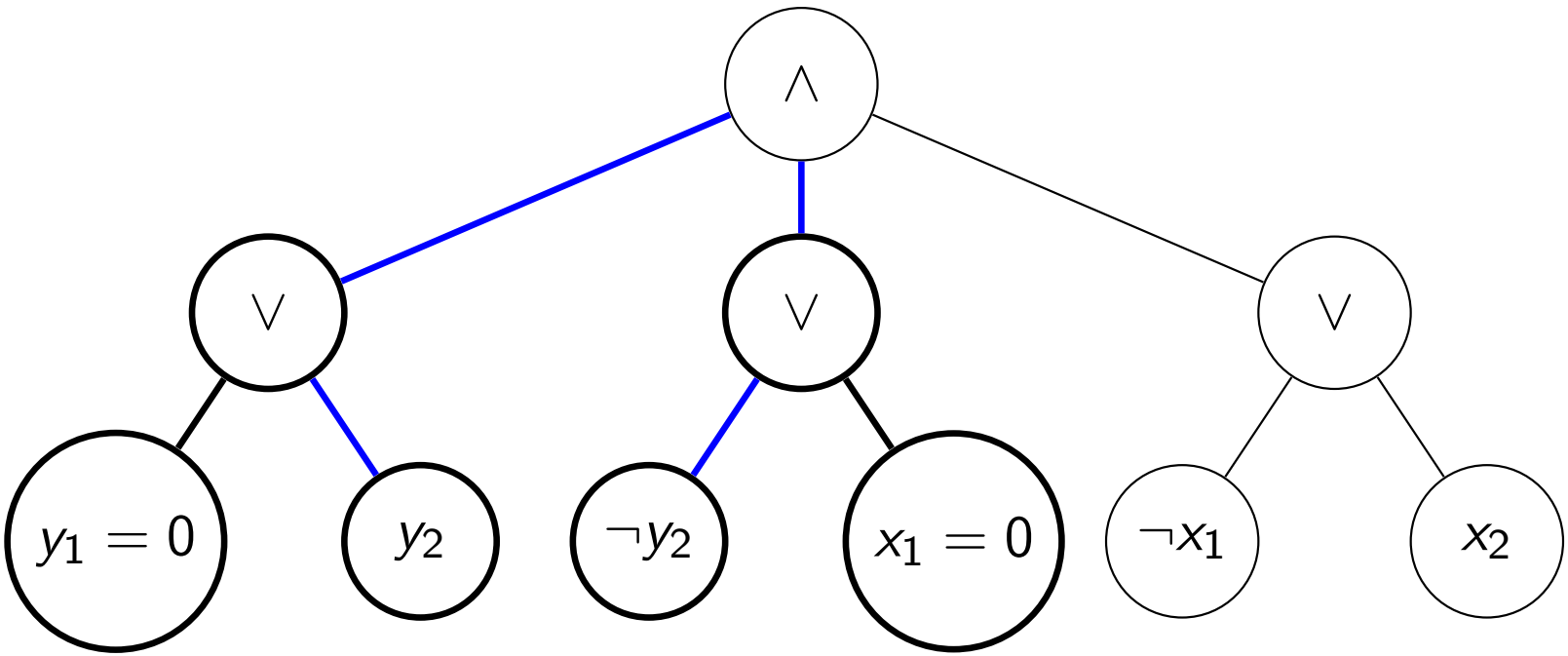


Representation of φ in SynNNF **w.r.t** (y_1, y_2)

Non-SynNNF: An Example

$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$

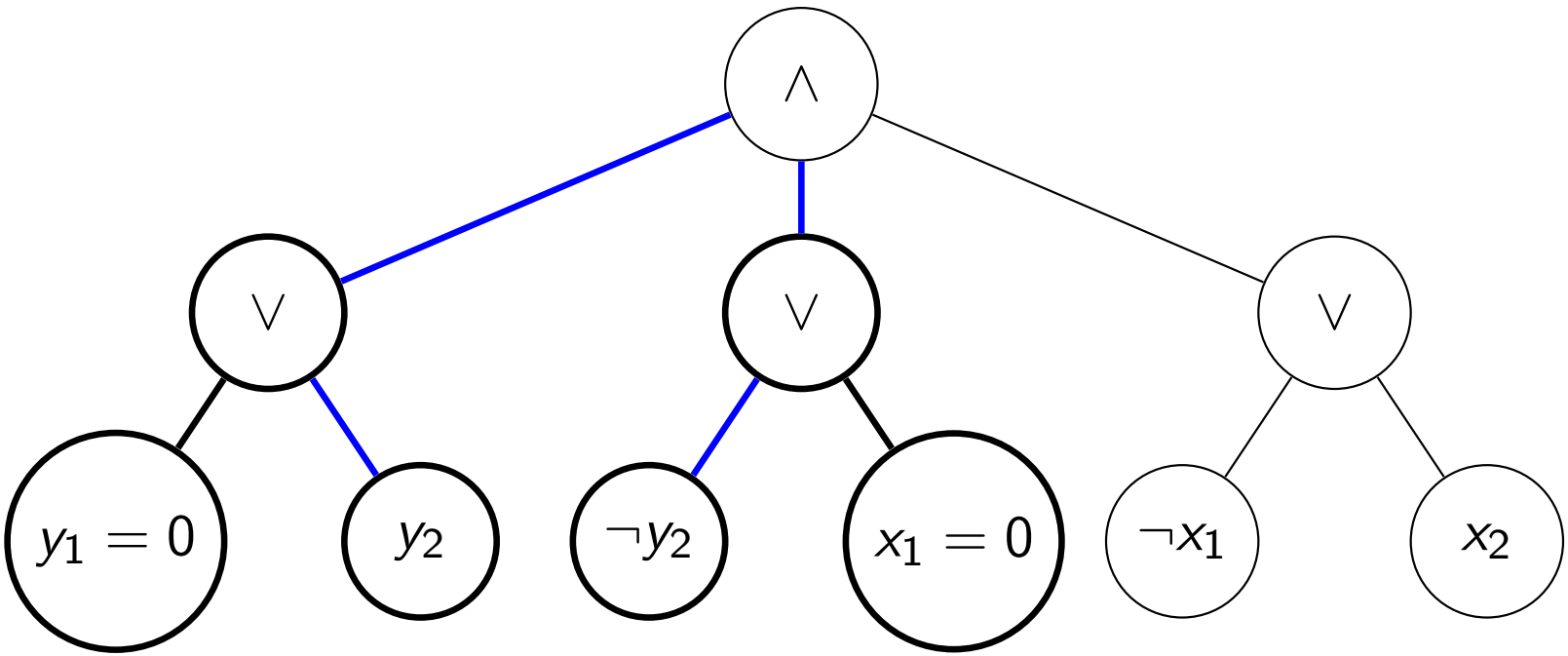
Output sequence: $(\underline{y_2}, y_1)$



Non-SynNNF: An Example

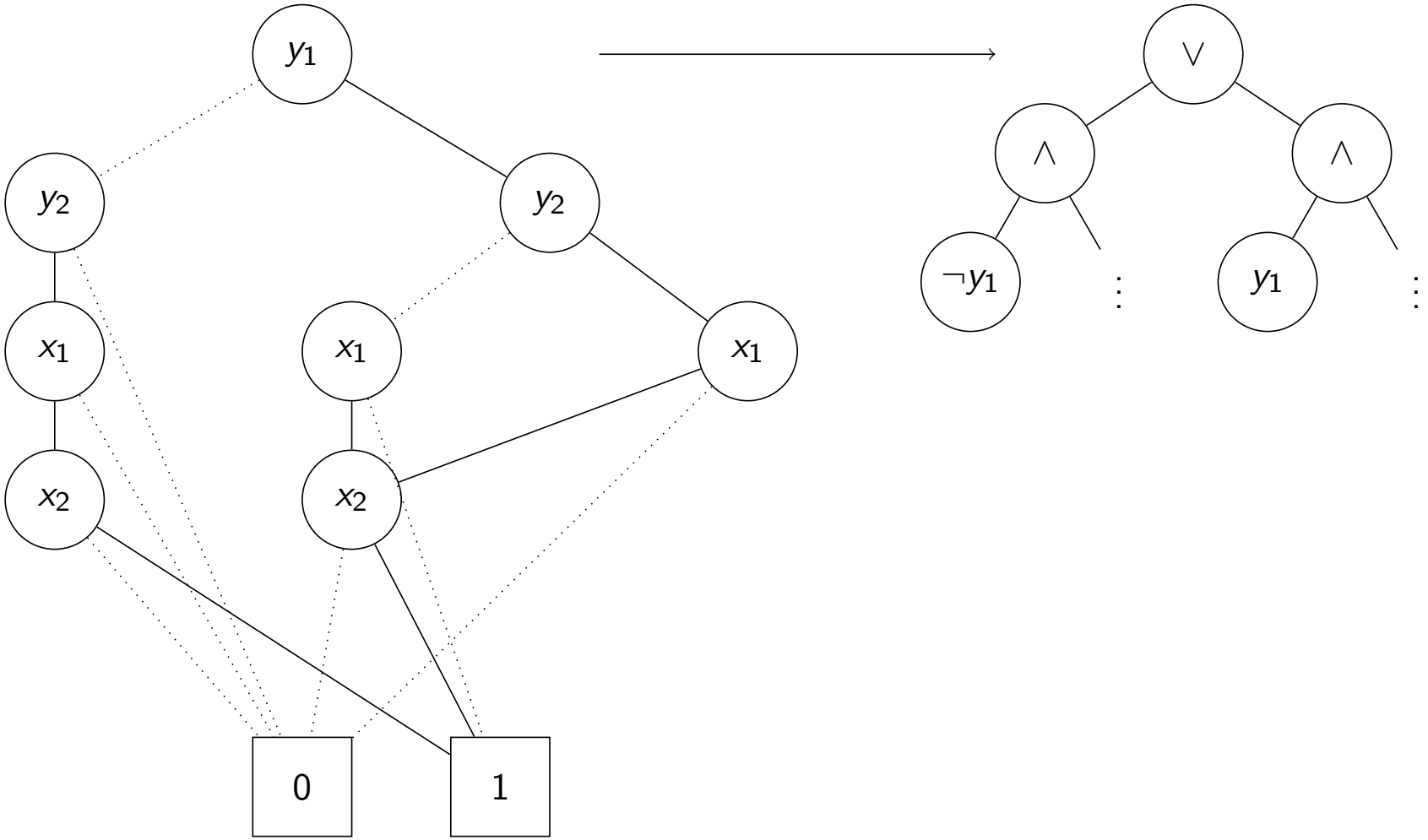
$\varphi(x_1, x_2, y_1, y_2) = (y_1 \vee y_2) \wedge (\neg y_2 \vee x_1) \wedge (\neg x_1 \vee x_2)$

Output sequence: $(\underline{y_2}, y_1)$

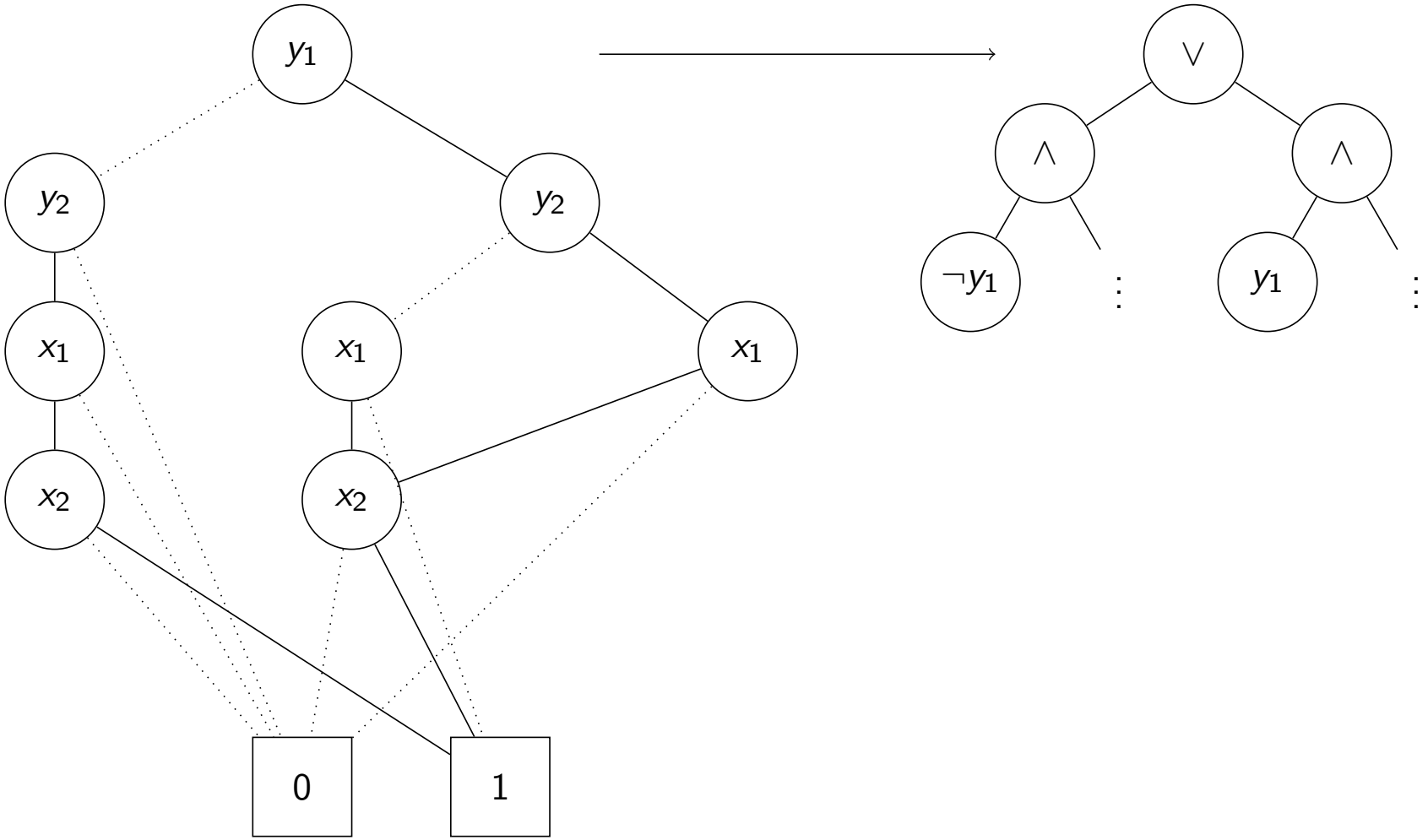


Representation of φ not in SynNNF **w.r.t** (y_2, y_1)

BDD and SynNNF



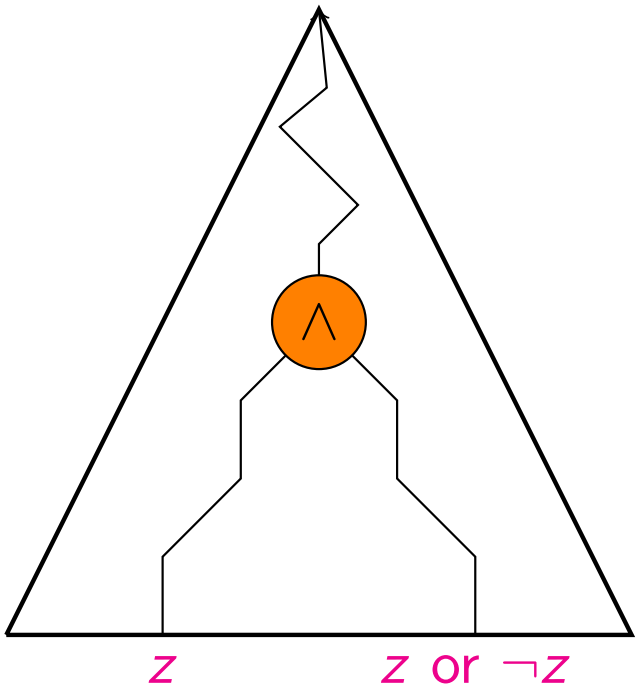
BDD and SynNNF



BDD $\rightarrow$ SynNNF in linear time for any output sequence and any BDD variable ordering.

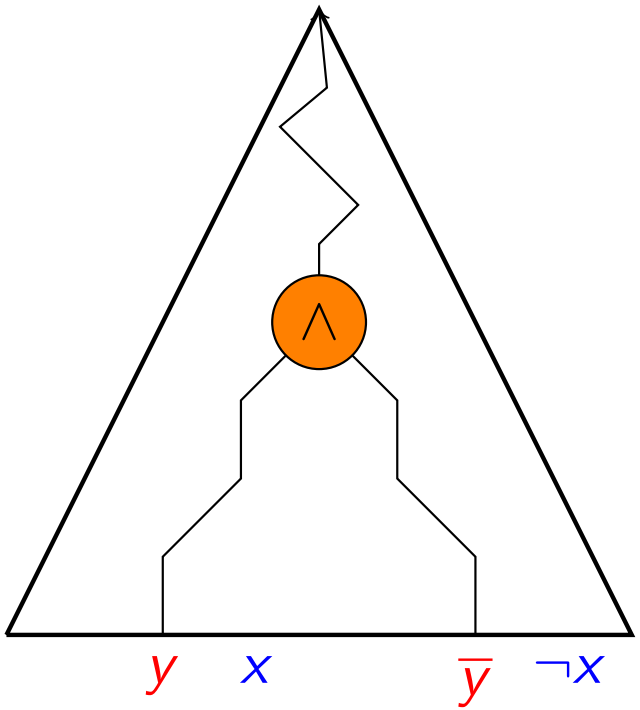
DNNF, wDNNF and SynNNF

φ in DNNF



Disallowed paths

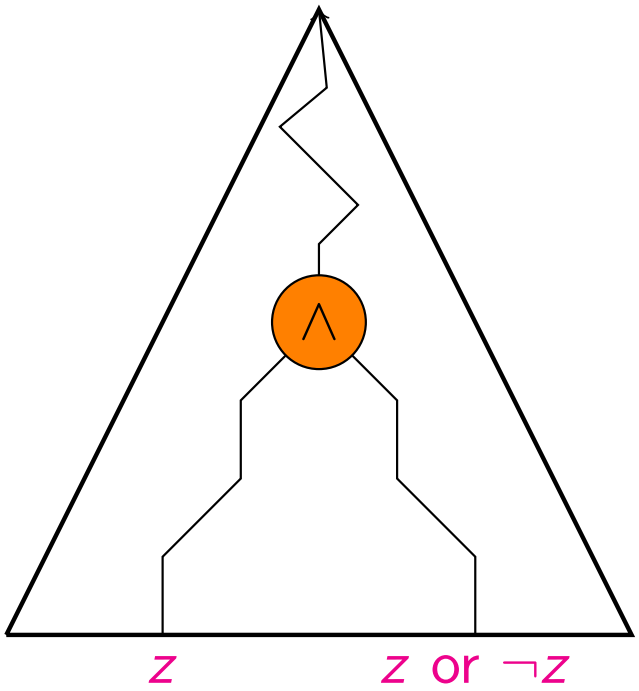
φ in wDNNF



Disallowed paths

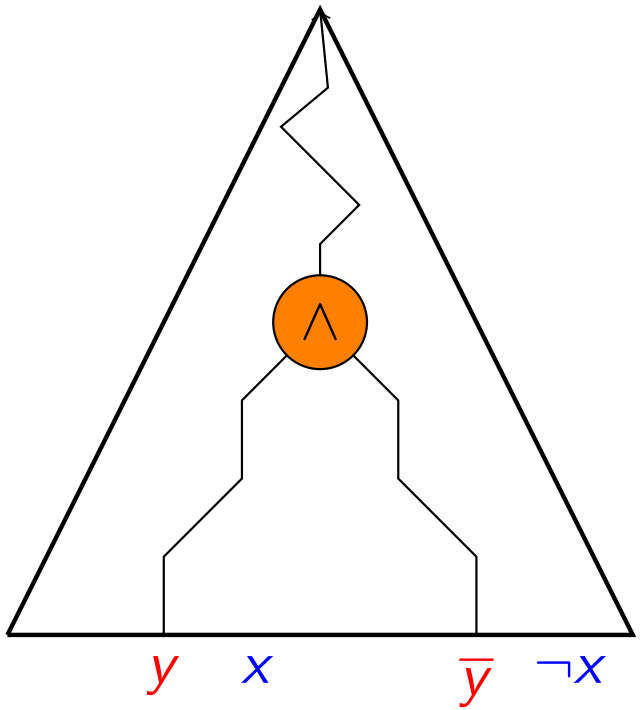
DNNF, wDNNF and SynNNF

φ in DNNF



Disallowed paths

φ in wDNNF



Disallowed paths

A specification in DNNF or wDNNF is already in SynNNF for any output sequence.

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

- Exponential-sized ROBDD/FBDD representations

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

- Exponential-sized ROBDD/FBDD representations
- Super-polynomial sized dDNNF representations, unless P = VNP

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

- Exponential-sized ROBDD/FBDD representations
- Super-polynomial sized dDNNF representations, unless P = VNP
- Super-polynomial sized wDNNF and DNNF representations unless P = NP

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

- Exponential-sized ROBDD/FBDD representations
- Super-polynomial sized dDNNF representations, unless P = VNP
- Super-polynomial sized wDNNF and DNNF representations unless P = NP

There exist poly-sized NNF representations that only admit super-polynomial sized SynNNF representations

- Unless the polynomial hierarchy collapses

SynNNF is relatively succinct

There exist polynomial sized SynNNF specifications that only admit

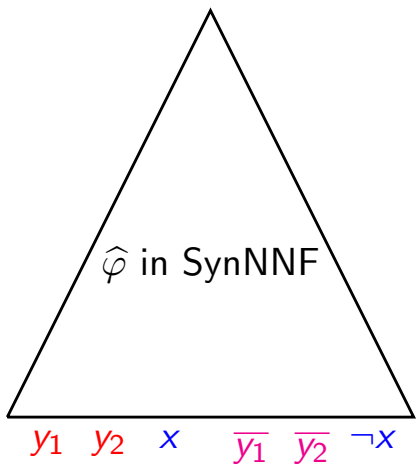
- Exponential-sized ROBDD/FBDD representations
- Super-polynomial sized dDNNF representations, unless P = VNP
- Super-polynomial sized wDNNF and DNNF representations unless P = NP

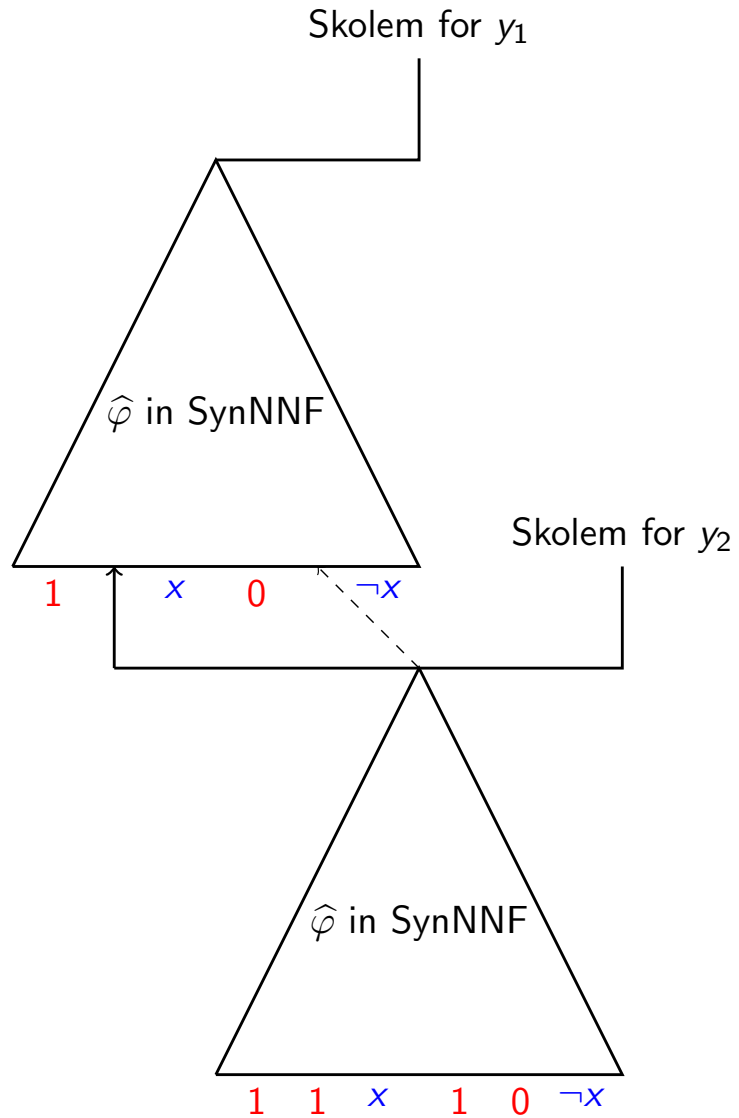
There exist poly-sized NNF representations that only admit super-polynomial sized SynNNF representations

- Unless the polynomial hierarchy collapses

$$\text{NNF} \sqsubset \text{SynNNF} \sqsubset \text{wDNNF} \sqsubset \text{DNNF} \sqsubset \text{dDNNF} \sqsubseteq \text{BDD}$$

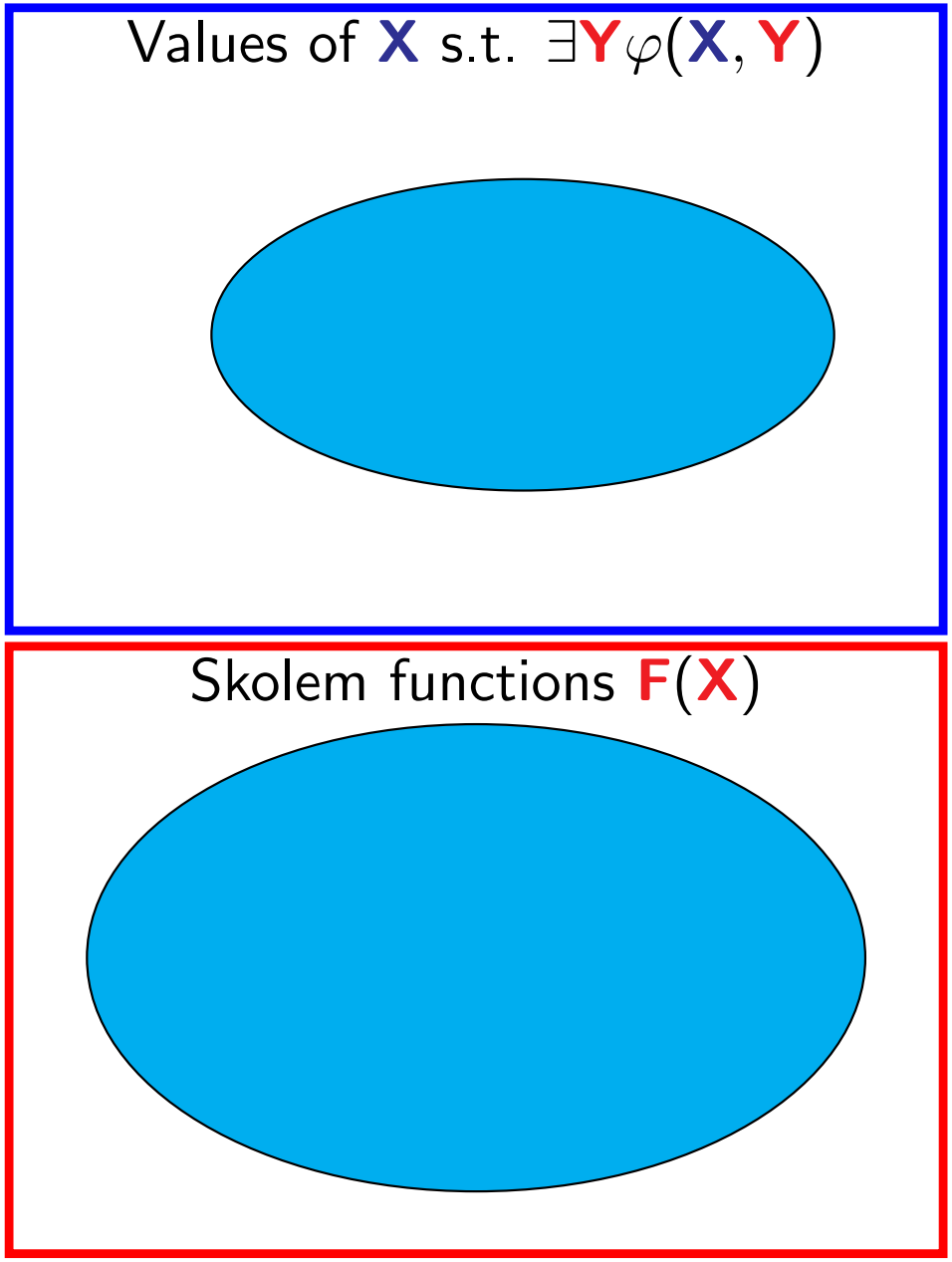
How does SynNNF help Skolem function synthesis?





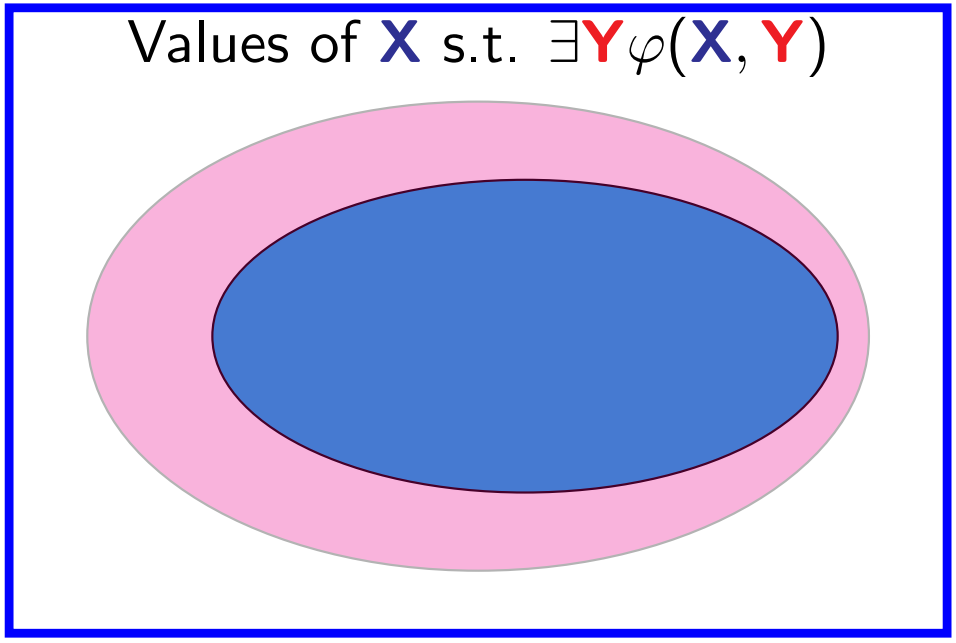
Synthesis: $m \times |\varphi|$ circuit size, $\mathcal{O}(m^2)$ additional wires.

Refinement w.r.t. synthesis

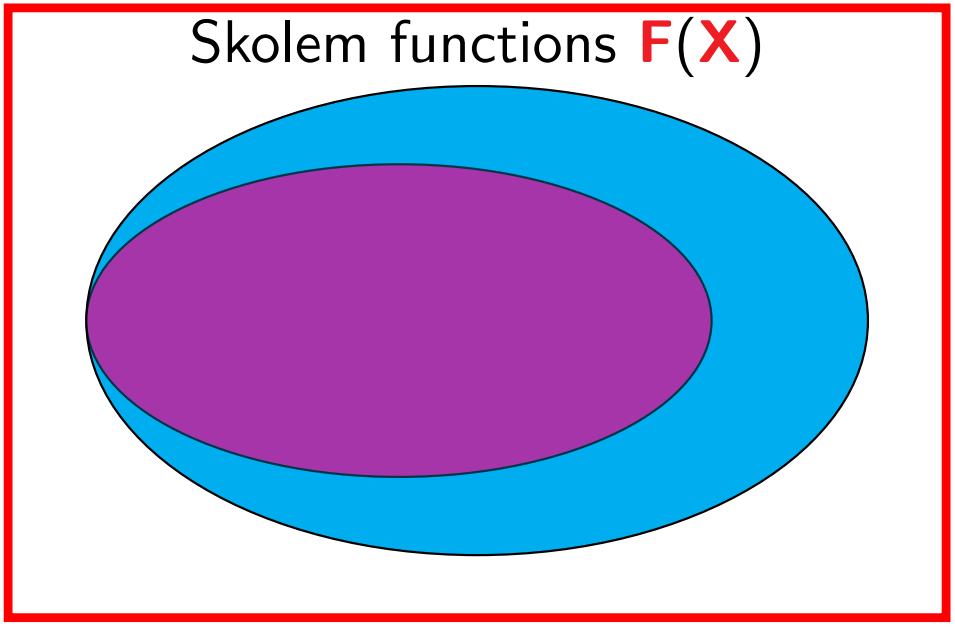


Given spec: $\varphi(\mathbf{X}, \mathbf{Y})$

Refinement w.r.t. synthesis



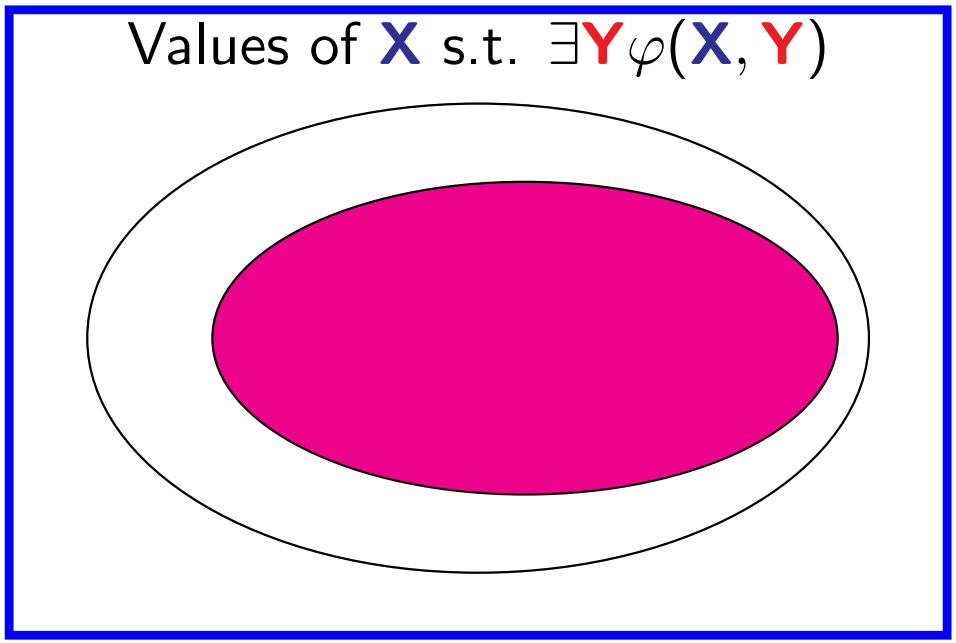
Given spec: $\varphi(\mathbf{X}, \mathbf{Y})$



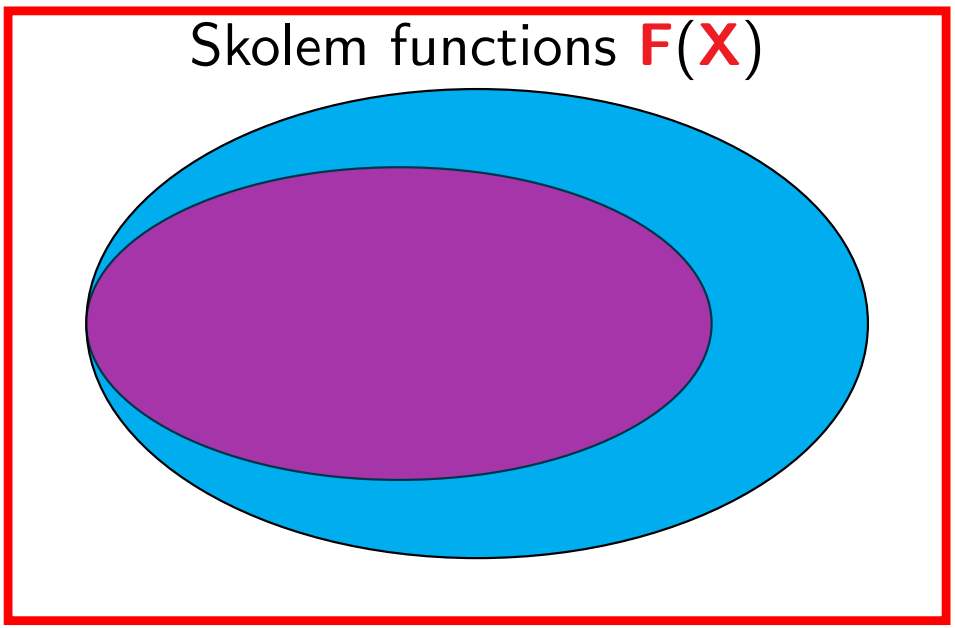
Refined spec: $\widetilde{\varphi}(\mathbf{X}, \mathbf{Y})$

$\widetilde{\varphi} \preceq_{syn} \varphi$

Refinement w.r.t. synthesis



Given spec: $\varphi(\mathbf{X}, \mathbf{Y})$



Strongly
Refined spec: $\widetilde{\varphi}(\mathbf{X}, \mathbf{Y})$
$\widetilde{\varphi} \preceq^{*}_{\mathit{syn}} \varphi$

Refinement w.r.t. synthesis

Lemma

If $\widetilde{F} \preceq_{syn} F$, every Skolem function vector for $\mathbf{Y}$ in $\widetilde{F}$ is also a Skolem function vector for $\mathbf{Y}$ in F.

Refinement w.r.t. synthesis

Lemma

If $\widetilde{F} \preceq_{syn} F$, every Skolem function vector for $\mathbf{Y}$ in $\widetilde{F}$ is also a Skolem function vector for $\mathbf{Y}$ in F.

Example: $(y_2 \wedge y_1) \preceq_{syn}$
$((\neg y_1 \vee y_2 \vee x_1) \wedge (y_1 \vee \neg y_2) \wedge (y_1 \vee \neg x_1) \wedge (y_2 \vee x_2))$

Putting it all together

Tool C2Syn:

- Input: φ in CNF (or AIG)
- Output: Refined $\widetilde{\varphi}$ in SynNNF

- Branches only on output variables
- Aggressively tries to refine whenever possible
- Details in our FMCAD 2019 paper

Experimental Results

Comparison of run-time with

- CADET [Rabe et al 2016]
- BFSS [Akshay et al 2018]
- BDD [BDD pipeline of BFSS]

Benchmarks: QBFEVAL 2018 and Factorization (408 total)

Benchmarks (Total)	Compiled By C2Syn			BDD compilation	Total in SynNNF
	Stage I	Stage II	Total		
QBFEVAL (402)	103	83	186	153	**283**
FA.QD (6)	0	6	6	6	**6**

Table: Compilation into SynNNF

Experimental Results

Bench mark	C2Syn vs CADET		C2Syn vs BFSS		C2Syn \ (CADET ∪ BFSS)
	C2Syn\ CADET	CADET\ C2Syn	C2Syn\ BFSS	BFSS\ C2Syn	
QBFEVAL	78	105	83	78	**75**
FA.QD	2	0	3	0	**2**

Table: Comparison Results of C2Syn

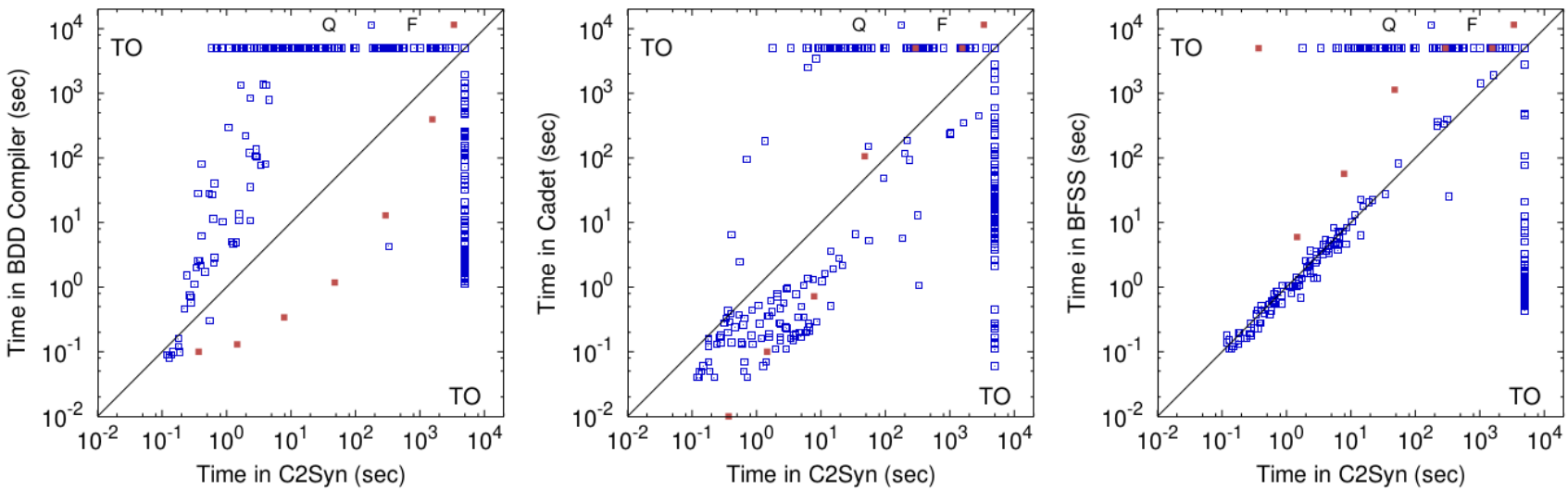


Fig. 1: Time comparisons: C2Syn vs BDD^{BFSS}, CADET, BFSS