# Synthesis of Boolean Functions

Markus N. Rabe

University of California at Berkeley

SAT+SMT School, Mysore
Dec 6-8, 2017

# Program Synthesis as an application of QBF

**Program sketching**: [Solar-Lezama'08]

Replace 'holes' (**??**) by an integer such that the assertion is satisfied for all inputs.

```
1 int abs(int x) {
2   int r;
3   if (x > ??) r =  x;
4   else        r = -x;
5   assert(r = |x|); // mathematical notation
6   return r;
7 }
```

An encoding in QBF:

$$\exists y \in \mathbb{B}^{32}\ \forall x \in \mathbb{B}^{32}\ \exists r \in \mathbb{B}^{32}.\ (x > y \rightarrow r = x)\ \wedge\ (x \leq y \rightarrow r = -x)\ \wedge\ r = |x|$$

**Observation:** The Skolem function (mapping $x$ to $r$) represents the program semantics.

# Beyond Certification - Program Synthesis Revisited

**Program sketching with expressions**:

Replace 'holes' (**???**) by an **expression** such that the assertion is satisfied for all inputs.

```
1 int abs(int x) {
2   int r;
3   if ( ??? ) r =  x;
4   else       r = -x;
5   assert(r = |x|); // mathematical notation
6   return r;
7 }
```

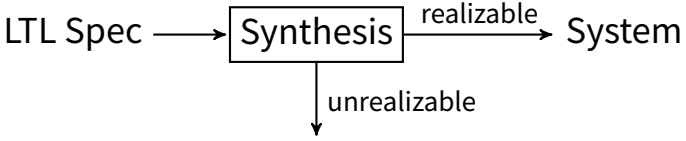
An encoding in QBF:

$$\forall x \in \mathbb{B}^{32}\ \exists c \in \mathbb{B},\ r \in \mathbb{B}^{32}.\ (c \to r = x) \wedge (\neg c \to r = -x) \wedge r = |x|$$

**The certificate for $c$ is the missing part of the implementation!**

# Bounded Synthesis of Reactive Systems

[Schewe, Finkbeiner'07]

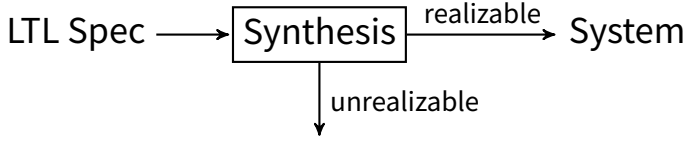



Applications: hardware bus systems, motion planning in robotics, …

# Bounded Synthesis of Reactive Systems

[Schewe, Finkbeiner'07]

$$\square(r_1 \rightarrow \bigcirc\Diamond g_1) \wedge$$
$$\square(r_2 \rightarrow \bigcirc\Diamond g_2) \wedge$$
$$\square\neg(g_1 \wedge g_2)$$

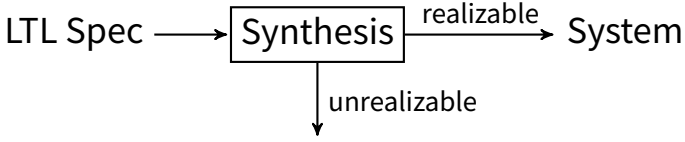


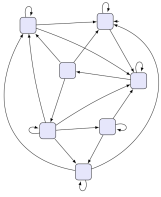

Applications: hardware bus systems, motion planning in robotics, …

# Bounded Synthesis of Reactive Systems

[Schewe, Finkbeiner'07]

$$\square(r_1 \rightarrow \bigcirc\Diamond g_1) \wedge$$
$$\square(r_2 \rightarrow \bigcirc\Diamond g_2) \wedge$$
$$\square\neg(g_1 \wedge g_2)$$



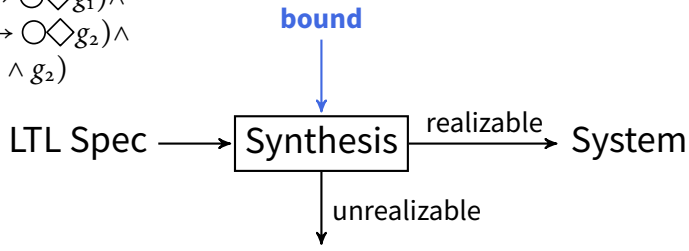



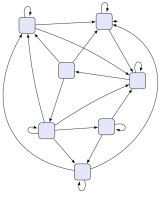
Applications: hardware bus systems, motion planning in robotics, …

# Bounded Synthesis of Reactive Systems

[Schewe, Finkbeiner'07]

$\square(r_1 \rightarrow \bigcirc\Diamond g_1) \wedge$
$\square(r_2 \rightarrow \bigcirc\Diamond g_2) \wedge$
$\square\neg(g_1 \wedge g_2)$



Applications: hardware bus systems, motion planning in robotics, …

In general, reactive synthesis needs automata-theoretic algorithms, but if we **bound** the number of states, we can reduce it to solving a constraint system.

# QBF encoding of Bounded Synthesis

[Faymonville et al.'17]

Given an LTL specification $\varphi$, find a system $S$ such that $\varphi \models S$. Let $Q$ be the states of the universal co-Büchi automaton of $\varphi$. Let $S$ have a fixed set of states $T$, a set of input variables $I$, and a set of output variables $O$.

$$\exists \{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$

$$\forall I$$

$$\exists \{\tau_{t,t'} \mid (t,t') \in T \times T\}$$

$$\exists \{o_t \mid o \in O, t \in T\}$$

$$\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigvee_{t' \in T} \tau_{t,t'}$$

$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left( \lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \left( \delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \left( \tau_{t,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

# QBF encoding of Bounded Synthesis

[Faymonville et al.'17]

Given an LTL specification $\varphi$, find a system $S$ such that $\varphi \models S$. Let $Q$ be the states of the universal co-Büchi automaton of $\varphi$. Let $S$ have a fixed set of states $T$, a set of input variables $I$, and a set of output variables $O$.

$$\exists \{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$
$$\forall I$$
$$\exists \{\tau_{t,t'} \mid (t,t') \in T \times T\}$$
$$\exists \{o_t \mid o \in O, t \in T\}$$
$$\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigvee_{t' \in T} \tau_{t,t'}$$
$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left( \lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \left( \delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \left( \tau_{t,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

In particular, the encoding has variables $o_t$ for every state $t \in T$. Their Skolem functions are the implementation of the system.

In practice, QBF solvers find small implementations.

# Invariant Synthesis

```
int multiply(int a, int b) { // assume a is positive
  int acc = 0;
  for(int i = 0; i < a; i++) {
    acc += b;
  }
  return acc;
}
```

Loop invariant: $(a - i) \cdot b + acc = a \cdot b$

# Invariant Synthesis

```
1 int multiply(int a, int b) { // assume a is positive
2   int acc = 0;
3   for(int i = 0; i < a; i++) {
4     acc += b;
5   }
6   return acc;
7 }
```

Loop invariant: $(a - i) \cdot b + acc = a \cdot b$

Find it automatically by solving this formula: $\exists inv : \mathbb{B}^{32\cdot 4} \to \mathbb{B}$.

$$\forall a, b.\ inv(a, b, 0, 0) \ \wedge \qquad \text{loop entry}$$

$$\forall a, b, i, acc.\ inv(a, b, i, acc) \ \wedge\ i = a \ \to\ acc = a \cdot b \ \wedge \qquad \text{loop exit}$$

$$\forall a, b, i, acc, a', b', i', acc'. \qquad \text{inductiveness}$$

$$a = a' \ \wedge\ b = b' \ \wedge\ i' = i + 1 \ \wedge\ acc' = acc + b \ \wedge inv(a, b, i, acc) \to inv(a', b', i', acc')$$

This is not a QBF! But can be expressed in DQBF, an extension of QBF.

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# A Performance Problem in Quantified Reasoning

```
int return_something_different(int y) {
    int x = ??; // choose an integer
    assert(x != y);
    return x;
}
```

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\ \forall y \in \mathbb{B}^{32}.\ x \neq y$

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\ \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\ \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\, \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

To apply QBF solvers, we need the formula to be in CNF.

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\ \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\ \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

To apply QBF solvers, we need the formula to be in CNF.

Naively applying the Tseitin transformation provides leads to a 3QBF:

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\, \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

To apply QBF solvers, we need the formula to be in CNF.

Naively applying the Tseitin transformation provides leads to a 3QBF:

$$\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}\, \exists t_1, \ldots, t_{32}.\ \left(\bigvee_i \neg t_i\right) \wedge \underbrace{\bigwedge_i (x_i \vee y_i \vee t_i) \wedge (\neg x_i \vee \neg y_i \vee t_i)}_{\bigwedge_i (x_i = y_i) \rightarrow t_i}$$

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\, \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

To apply QBF solvers, we need the formula to be in CNF.

Naively applying the Tseitin transformation provides leads to a 3QBF:

$$\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}\, \exists t_1, \ldots, t_{32}.\ \left(\bigvee_i \neg t_i\right) \wedge \underbrace{\bigwedge_i (x_i \vee y_i \vee t_i) \wedge (\neg x_i \vee \neg y_i \vee t_i)}_{\bigwedge_i (x_i = y_i) \rightarrow t_i}$$

# A Performance Problem in Quantified Reasoning

```
1 int return_something_different(int y) {
2     int x = ??; // choose an integer
3     assert(x != y);
4     return x;
5 }
```

As quantified bitvector formula: $\exists x \in \mathbb{B}^{32}\, \forall y \in \mathbb{B}^{32}.\ x \neq y$

As quantified Boolean formula: $\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}.\ \bigvee_i x_i \neq y_i$

To apply QBF solvers, we need the formula to be in CNF.

Naively applying the Tseitin transformation provides leads to a 3QBF:

$$\exists x_1, \ldots, x_{32}\, \forall y_1, \ldots, y_{32}\, \exists t_1, \ldots, t_{32}.\ (\bigvee_i \neg t_i) \wedge \underbrace{\bigwedge_i (x_i \vee y_i \vee t_i) \wedge (\neg x_i \vee \neg y_i \vee t_i)}_{\bigwedge_i (x_i = y_i) \rightarrow t_i}$$

Negating first leads to a QCNF in 2QBF: $\forall x_1, \ldots, x_{32}\, \exists y_1, \ldots, y_{32}.\ \bigwedge_i x_i = y_i$

$\forall x_1, \ldots, x_{32}\, \exists y_1, \ldots, y_{32}.\ \bigwedge_i (x_i \vee \neg y_i) \wedge (\neg x_i \vee y_i)$

# $\forall x_1, \ldots, x_{32}\ \exists y_1, \ldots, y_{32}\ .\ \bigwedge_i (x_i \vee \neg y_i) \wedge (\neg x_i \vee y_i)$ in QDIMACS

```
p cnf 64 64
a 1 2 [...] 31 32 0
e 33 34 [...] 63 64 0
1 -33 0
-1 33 0
2 -34 0
-2 34 0
[...]
30 -62 0
-30 62 0
31 -63 0
-31 63 0
32 -64 0
```

64 variables and 64 clauses is **tiny** for QBF solvers and the formula is trivial, yet most QBF solvers have problems solving it without preprocessing.

**DEMO**

# A Closer Look at the Problem

What do QCDCL/CEGAR solvers for QBF do for $\forall X \exists Y.\ \varphi(X, Y)$?

1. Start with formula $\psi := \mathit{true}$ over variables $X$.
2. Search a solution $x$ for $\psi$.
3. If there is no such assignment, terminate with *true*.
4. Search an assignment $y$ to $\varphi(x, Y)$.
5. If there is no such assignment, terminate with *false*.
6. Exclude all assignments to $X$ for which $\varphi(X, y)$; i.e. $\psi' := \psi \wedge \neg\varphi(X, y)$
7. Go back to step 2.

# A Closer Look at the Problem

What do QCDCL/CEGAR solvers for QBF do for $\forall X \exists Y.\ \varphi(X, Y)$?

1. Start with formula $\psi := \mathit{true}$ over variables $X$.
2. Search a solution $x$ for $\psi$.
3. If there is no such assignment, terminate with *true*.
4. Search an assignment $y$ to $\varphi(x, Y)$.
5. If there is no such assignment, terminate with *false*.
6. Exclude all assignments to $X$ for which $\varphi(X, y)$; i.e. $\psi' := \psi \wedge \neg\varphi(X, y)$
7. Go back to step 2.

Take any Boolean function $f$, consider the formula $\forall x \exists y.\ f(x) = y$.

# A Closer Look at the Problem

What do QCDCL/CEGAR solvers for QBF do for $\forall X \exists Y.\ \varphi(X, Y)$?

1. Start with formula $\psi := \mathit{true}$ over variables $X$.
2. Search a solution $x$ for $\psi$.
3. If there is no such assignment, terminate with *true*.
4. Search an assignment $y$ to $\varphi(x, Y)$.
5. If there is no such assignment, terminate with *false*.
6. Exclude all assignments to $X$ for which $\varphi(X, y)$; i.e. $\psi' := \psi \wedge \neg\varphi(X, y)$
7. Go back to step 2.

Take any Boolean function $f$, consider the formula $\forall x \exists y.\ f(x) = y$.

Number of iterations is the size of the co-domain of $f$. But the formula is trivial; the certificate is $f$.

# A Closer Look at the Problem

What do QCDCL/CEGAR solvers for QBF do for $\forall X\ \exists Y.\ \varphi(X, Y)$?

1. Start with formula $\psi := \mathit{true}$ over variables $X$.
2. Search a solution $x$ for $\psi$.
3. If there is no such assignment, terminate with *true*.
4. Search an assignment $y$ to $\varphi(x, Y)$.
5. If there is no such assignment, terminate with *false*.
6. Exclude all assignments to $X$ for which $\varphi(X, y)$; i.e. $\psi' := \psi \wedge \neg\varphi(X, y)$
7. Go back to step 2.

Take any Boolean function $f$, consider the formula $\forall x\ \exists y.\ f(x) = y$.

Number of iterations is the size of the co-domain of $f$. But the formula is trivial; the certificate is $f$.

**Corollary:**
Solving the formula $\forall x_1, \ldots, x_n\ \exists y_1, \ldots, y_n.\ \bigwedge_i f(x_i) = y_i$ takes $|coDom(f)|^n$ iterations.

# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}:\ ?} \wedge$$

$$(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)$$

# A Different Approach - Example

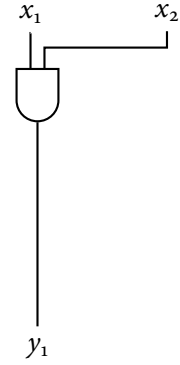
$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(\neg x_1 \rightarrow \neg y_1) \wedge (\neg x_2 \rightarrow \neg y_1) \wedge (x_1 \wedge x_2 \rightarrow y_1)}_{f_{y_1}:} \wedge$$

$$(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)$$

# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

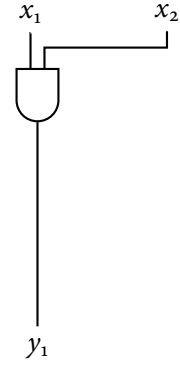
$$(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)$$



# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

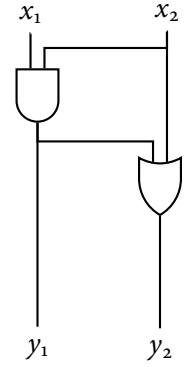
$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon ?}$$



# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)}$$



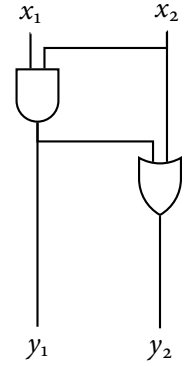
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \ \wedge$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



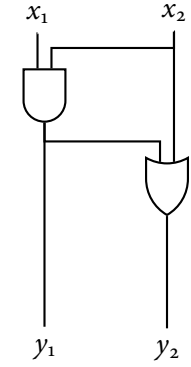
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge \underbrace{(\neg y_1 \vee \neg y_2 \vee y_3)}_{\text{decision clause}} \wedge$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



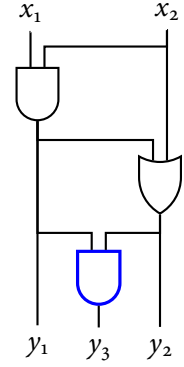
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2) \wedge f_{y_2}(x_1, x_2)} \wedge$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



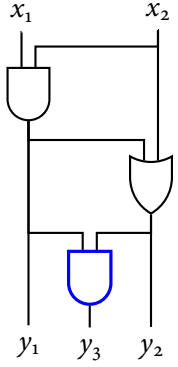
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2) \wedge f_{y_2}(x_1, x_2)} \wedge$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)}_{\text{conflict for } x_1 \wedge x_2}$$



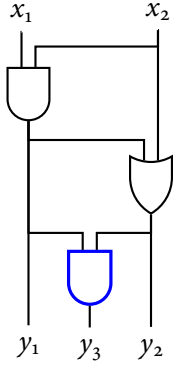
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge (x_2 \vee y_1 \vee \neg y_2) \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2) \wedge f_{y_2}(x_1, x_2)} \wedge$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)}_{\text{conflict for } x_1 \wedge x_2}$$



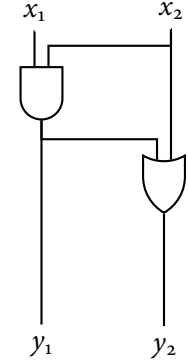
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge \underbrace{(\neg y_1 \vee \neg y_3)}_{\text{learnt clause}}$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



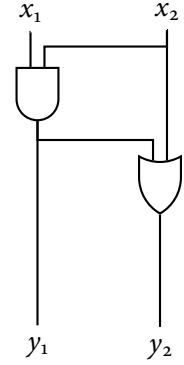
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



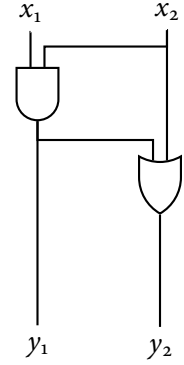
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$



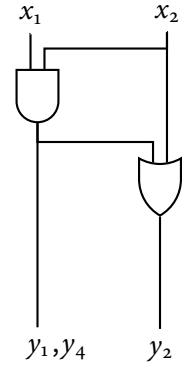
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)$$



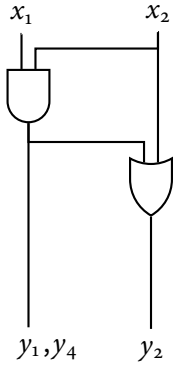
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)}_{f_{y_4}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2)}$$



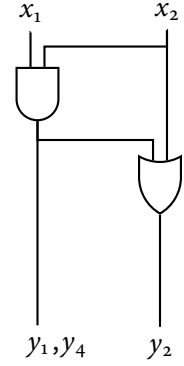
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)}_{f_{y_4}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2)}$$




Skolem function has 3 distinct outputs; QCDCL/CEGAR needs 3 iterations.

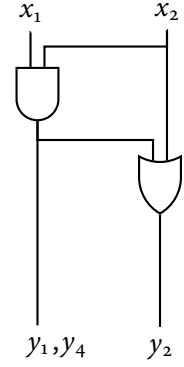
# A Different Approach - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{f_{y_1}\colon (x_1, x_2) \mapsto x_1 \wedge x_2} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{f_{y_2}\colon (x_1, x_2) \mapsto x_2 \vee f_{y_1}(x_1, x_2)} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{f_{y_3}\colon (x_1, x_2) \mapsto 0}$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)}_{f_{y_4}\colon (x_1, x_2) \mapsto f_{y_1}(x_1, x_2)}$$




Skolem function has 3 distinct outputs; QCDCL/CEGAR needs 3 iterations.
Exploited **functional dependencies** to derive avoid enumeration of outputs.

# Incremental Determinization [R., Seshia'16]

CDCL in the function space:

| | SAT $\exists Y.\ \varphi$ | QBF $\forall X.\exists Y.\ \varphi$ |
|---|---|---|
| Partial assignment | $Y \to \mathbb{B} \cup \{\top\}$ | $Y \to ((X \to \mathbb{B}) \to \mathbb{B}) \cup \{\top\}$ |
| Decision | literal | constraints/function |
| Conflict | propagation of $y$ and $\neg y$ | $\exists X$. propagation of $y$ and $\neg y$ |
| Learning | clause | clause |
| **Propagation** | unit propagation | **unique Skolem function** |

# Incremental Determinization [R., Seshia'16]

CDCL in the function space:

| | SAT $\exists Y.\ \varphi$ | QBF $\forall X.\exists Y.\ \varphi$ |
|---|---|---|
| Partial assignment | $Y \to \mathbb{B} \cup \{\top\}$ | $Y \to ((X \to \mathbb{B}) \to \mathbb{B}) \cup \{\top\}$ |
| Decision | literal | constraints/function |
| Conflict | propagation of $y$ and $\neg y$ | $\exists X.$ propagation of $y$ and $\neg y$ |
| Learning | clause | clause |
| **Propagation** | unit propagation | **unique Skolem function** |

Rest of part 1 of this tutorial:

- Propagation, Decisions, Conflict Analysis
- Termination

# Incremental Determinization - An Overview

1: **procedure** IncrementalDeterminization($\forall X.\exists Y.\varphi$)
2: $dlvl \leftarrow 0$; $D \leftarrow \varnothing$
3: **while** true **do**
4: $D, \varphi, conflict, \mathbf{x} \leftarrow$ Propagate($\forall X.\exists Y.\varphi, D, dlvl$)
5: **if** $conflict$ **then**
6: $c \leftarrow$ AnalyzeConflict($\forall X.\exists Y.\ \varphi, \mathbf{x}, D, dlvl$)
7: **if** $c$ only contains variables in $X$ **then**
8: **return** false
9: $dlvl \leftarrow$ (maximal decision level in $c$) $- 1$
10: $\varphi, D \leftarrow$ Backtrack($\varphi, D, dlvl$)
11: $\varphi \leftarrow \varphi \wedge c$
12: **else**
13: **if** $D = Y$ **then**
14: **return** true
15: $v \leftarrow$ PickVar($Y \setminus D$)
16: $dlvl \leftarrow dlvl + 1$
17: $\varphi \leftarrow \varphi \wedge$ Decision($v, \varphi, D$)

# Incremental Determinization - An Overview

```
1:  procedure IncrementalDeterminization(∀X.∃Y.φ)
2:      dlvl ← 0;  D ← ∅
3:      while true do
4:          D, φ, conflict, x ← Propagate(∀X.∃Y.φ, D, dlvl)
5:          if conflict then
6:              c ← AnalyzeConflict(∀X.∃Y. φ, x, D, dlvl)
7:              if c only contains variables in X then
8:                  return false
9:              dlvl ← (maximal decision level in c) − 1
10:             φ, D ← Backtrack(φ, D, dlvl)
11:             φ ← φ ∧ c
12:         else
13:             if D = Y then
14:                 return true
15:             v ← PickVar(Y ∖ D)
16:             dlvl ← dlvl + 1
17:             φ ← φ ∧ Decision(v, φ, D)
```

**Main data structure:** a set $D$ of variables.

**Invariant:** For every assignment to $X$ the clauses that contain only variables in $D$ admit exactly one assignment to $D$, which can be computed by Boolean constraint propagation.

# Incremental Determinization - An Overview

```
1:  procedure IncrementalDeterminization(∀X.∃Y.φ)
2:      dlvl ← 0;  D ← ∅
3:      while true do
4:          D, φ, conflict, x ← Propagate(∀X.∃Y.φ, D, dlvl)
5:          if conflict then
6:              c ← AnalyzeConflict(∀X.∃Y. φ, x, D, dlvl)
7:              if c only contains variables in X then
8:                  return false
9:              dlvl ← (maximal decision level in c) − 1
10:             φ, D ← Backtrack(φ, D, dlvl)
11:             φ ← φ ∧ c
12:         else
13:             if D = Y then
14:                 return true
15:             v ← PickVar(Y ∖ D)
16:             dlvl ← dlvl + 1
17:             φ ← φ ∧ Decision(v, φ, D)
```

**Main data structure:** a set $D$ of variables.

**Invariant:** For every assignment to $X$ the clauses that contain only variables in $D$ admit exactly one assignment to $D$, which can be computed by Boolean constraint propagation.

Next up:

1. Propagate
2. Decision
3. AnalyzeConflict

# Unique consequences, unique Skolem functions

Given $\forall X.\exists Y.\varphi$ and a set $D \subseteq Y$ of variables that have a Skolem function already.

# Unique consequences, unique Skolem functions

Given $\forall X.\exists Y.\varphi$ and a set $D \subseteq Y$ of variables that have a Skolem function already.

Clause $(l_1 \vee \cdots \vee l_n)$ has **unique consequence** $y = var(l_n)$ if

$$y \in Y \setminus D, \text{ and } var(l_i) \in D \cup X \text{ for } i < n.$$

$\approx$ when all other existentials in the clause have a Skolem function already.

# Unique consequences, unique Skolem functions

Given $\forall X.\exists Y.\varphi$ and a set $D \subseteq Y$ of variables that have a Skolem function already.

Clause $(l_1 \vee \cdots \vee l_n)$ has **unique consequence** $y = var(l_n)$ if

$$y \in Y \setminus D, \text{ and } var(l_i) \in D \cup X \text{ for } i < n.$$

$\approx$ when all other existentials in the clause have a Skolem function already.

Variable $v$ has a **unique Skolem function** if $\forall X, D.\ \mathcal{D} \rightarrow \exists! v.\ \bigwedge UC_v$

# Unique consequences, unique Skolem functions

Given $\forall X.\exists Y.\varphi$ and a set $D \subseteq Y$ of variables that have a Skolem function already.

Clause $(l_1 \vee \cdots \vee l_n)$ has **unique consequence** $y = var(l_n)$ if

$$y \in Y \setminus D, \text{ and } var(l_i) \in D \cup X \text{ for } i < n.$$

$\approx$ when all other existentials in the clause have a Skolem function already.

Variable $v$ has a **unique Skolem function** if $\forall X, D.\ \mathcal{D} \rightarrow \exists!v.\ \bigwedge UC_v$

Two SAT checks:

- Functional: $\mathcal{D} \wedge \bigwedge (UC_v \setminus \{v, \neg v\})$
- Conflicted: $\mathcal{D} \wedge \bigvee \neg(UC_v[1/v]) \wedge \bigvee \neg(UC_v[0/v])$

# Propagation

1. Check all clauses for a unique consequence
2. Check all variables occurring as a unique consequence for a unique Skolem function.
3. Add variables with a unique Skolem function to $D$ and update the set of unique consequences.

# Propagation

1. Check all clauses for a unique consequence
2. Check all variables occurring as a unique consequence for a unique Skolem function.
3. Add variables with a unique Skolem function to $D$ and update the set of unique consequences.

Extends $D$ and maintains the **invariant:** For every assignment to $X$ the clauses that contain only variables in $D$ admit exactly one assignment to $D$, which can be computed by Boolean constraint propagation.

# Propagation

1. Check all clauses for a unique consequence
2. Check all variables occurring as a unique consequence for a unique Skolem function.
3. Add variables with a unique Skolem function to $D$ and update the set of unique consequences.

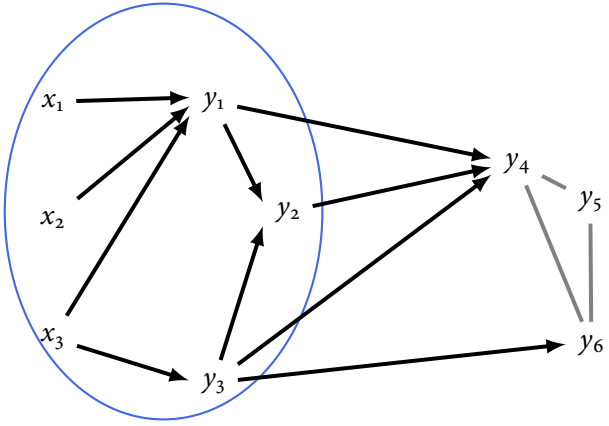
Extends $D$ and maintains the **invariant:** For every assignment to $X$ the clauses that contain only variables in $D$ admit exactly one assignment to $D$, which can be computed by Boolean constraint propagation.

Propagation alone solves $\forall x_1, \ldots, x_{32}\, \exists y_1, \ldots, y_{32}\, .\ \bigwedge_i (x_i \vee \neg y_i) \wedge (\neg x_i \vee y_i)$.

**DEMO**

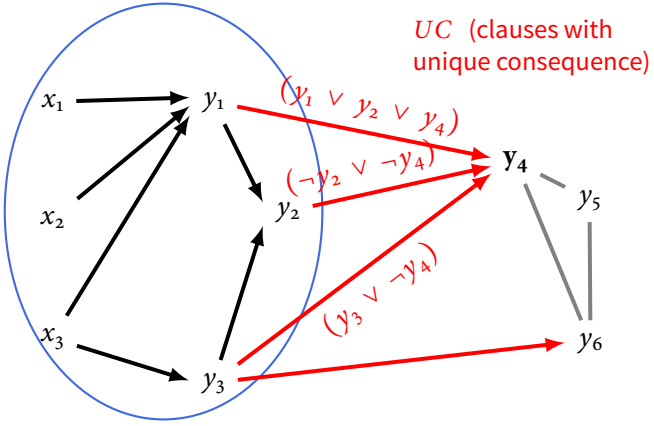
# Unique Consequences and the Partial Function View

D (variables with
unique Skolem function)



# Unique Consequences and the Partial Function View

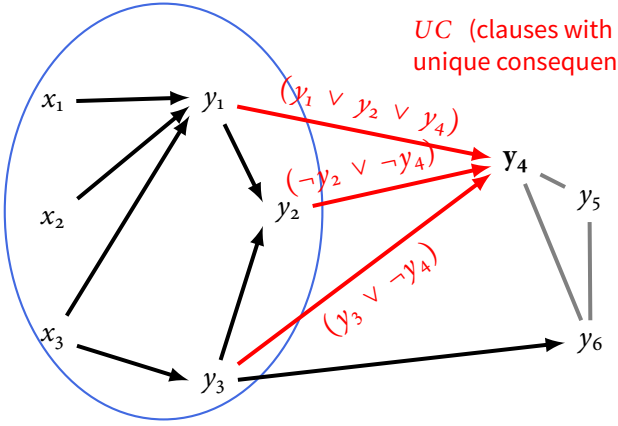
D (variables with unique Skolem function)



# Unique Consequences and the Partial Function View

D (variables with unique Skolem function)

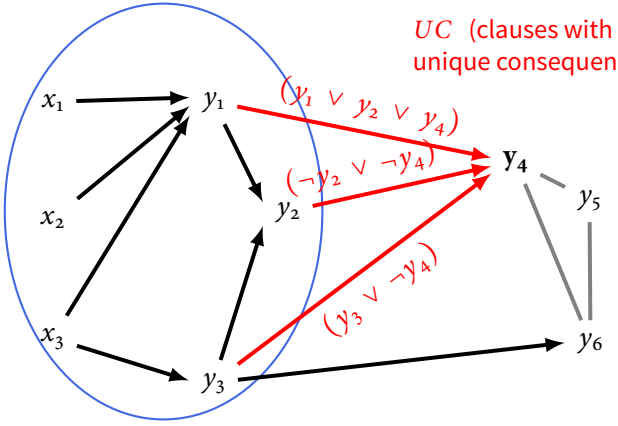
$UC$ (clauses with unique consequence)




| Condition | $y_4$ |
|---|---|
| $\neg y_1 \wedge \neg y_2$ | 1 |
| | ?? |
| $y_2$ | 0 |
| $\neg y_3$ | 0 |

# Unique Consequences and the Partial Function View

D (variables with unique Skolem function)




| Condition | $\mathbf{y_4}$ |
|---|---|
| $\neg y_1 \wedge \neg y_2$ | 1 |
| | ?? |
| $y_2$ | 0 |
| $\neg y_3$ | 0 |

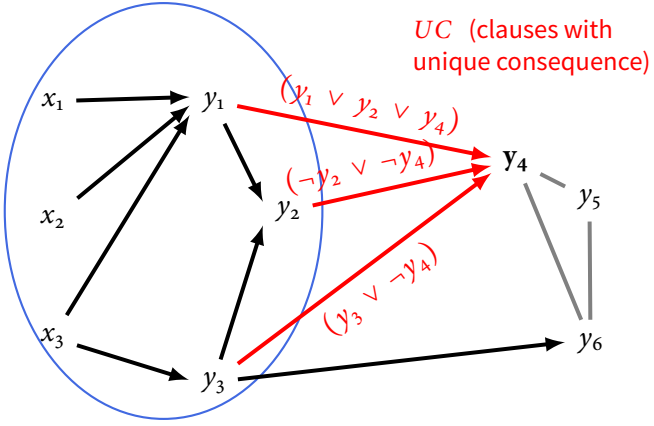
Functional: $?? \stackrel{?}{=} \varnothing$

Conflict: $0 \cap 1 \stackrel{?}{\neq} \varnothing$

Decision:
assign ?? to 0 or 1

# Unique Consequences and the Partial Function View

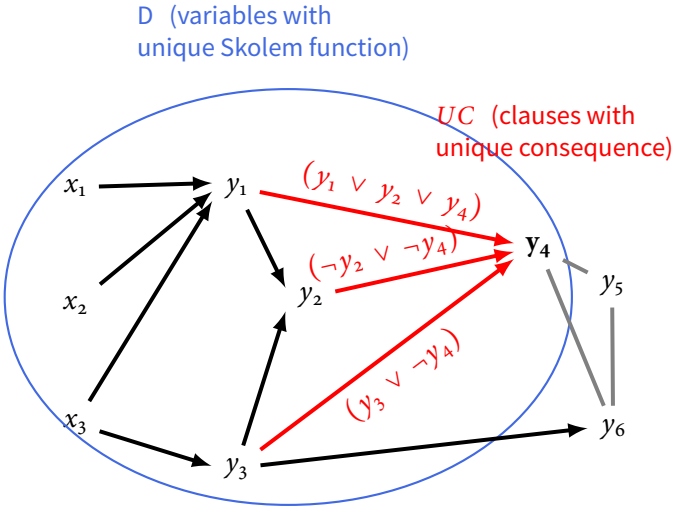
D (variables with unique Skolem function)




Assume $y_4$ is functional and not conflicted.

$\Rightarrow$ Can propagate $y_4$

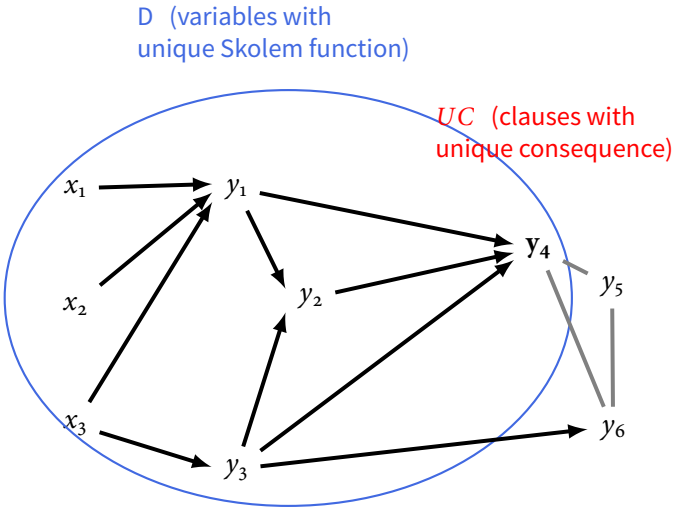
# Unique Consequences and the Partial Function View




Assume $y_4$ is functional and not conflicted.

$\Rightarrow$ Can propagate $y_4$

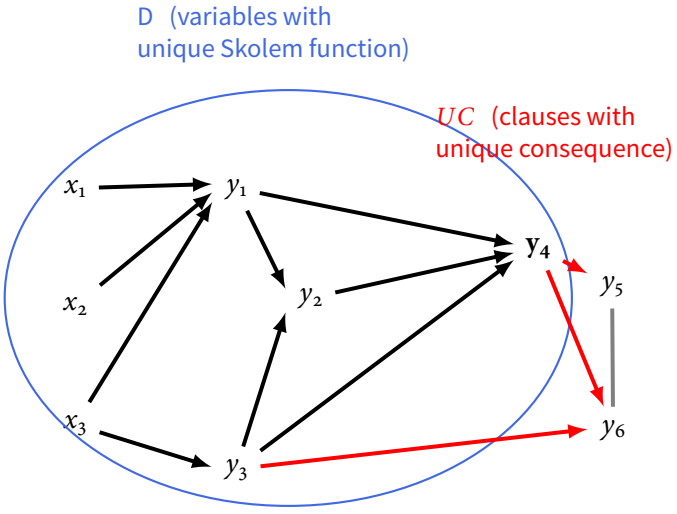
# Unique Consequences and the Partial Function View




Assume $y_4$ is functional and not conflicted.

$\Rightarrow$ Can propagate $y_4$

# Unique Consequences and the Partial Function View




Assume $y_4$ is functional and not conflicted.

$\Rightarrow$ Can propagate $y_4$

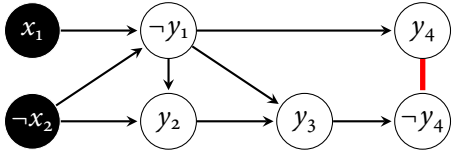
Get new clauses with unique consequence.

# Implication Graphs

**Implication graphs in CDCL for SAT (and in QCDCL):**

1. Add a node for every decision.
2. Add propagation edges: If $l^*$ was propagated (with Boolean constraint propagation) by a clause $(l_1 \vee \cdots \vee l_n \vee l*)$ add an edge $l_i \longrightarrow l'$ for every $i$.
3. A conflict is represented if for some variable both literals occur.

Example: $(x_1 \vee \neg x_2 \vee \neg y_1) \wedge (\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2 \vee y_3) \wedge (y_3 \vee \neg y_4) \wedge (\neg y_1 \vee y_4)$

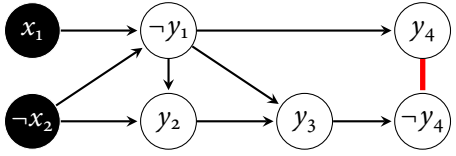


# Implication Graphs

**Implication graphs in CDCL for SAT (and in QCDCL):**

1. Add a node for every decision.
2. Add propagation edges: If $l^*$ was propagated (with Boolean constraint propagation) by a clause $(l_1 \vee \cdots \vee l_n \vee l*)$ add an edge $l_i \longrightarrow l'$ for every $i$.
3. A conflict is represented if for some variable both literals occur.

Example: $(x_1 \vee \neg x_2 \vee \neg y_1) \wedge (\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2 \vee y_3) \wedge (y_3 \vee \neg y_4) \wedge (\neg y_1 \vee y_4)$



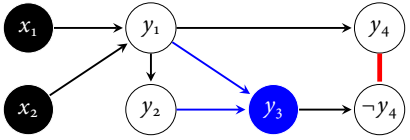

Lifting implication graphs to functions? Not necessary!

- We have the assignment for the universal variables.

# Conflict Analysis in Incremental Determinization

**How do decisions and assignments to the universal variables interact?**

- Propagating the universals assigns all variables in $D$ and the conflicted variable.

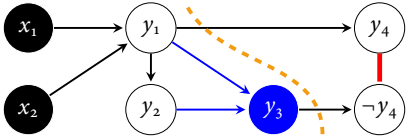



$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\quad (x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1) \wedge$$
$$(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2) \wedge$$
$$(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3) \wedge$$
$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$

# Conflict Analysis in Incremental Determinization

**How do decisions and assignments to the universal variables interact?**

- Propagating the universals assigns all variables in $D$ and the conflicted variable.




$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\quad (x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1) \wedge$$
$$(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2) \wedge$$
$$(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_2 \vee y_3) \wedge$$
$$(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4)$$

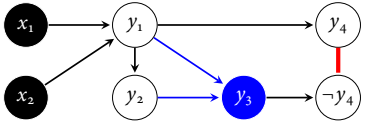
Find a cut separating the conflict from the universals **and the decision variables**.

The learnt clause is the negation of the conjunction of the origins of all cut edges.

Here: $(\neg y_1 \vee \neg y_3)$

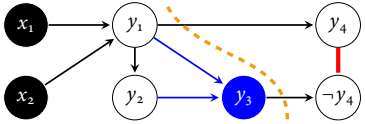
# Termination

**Important property:**
For the same decisions and universal assignment, adding the conflict clause and removing all clauses right of the cut, would still result in a conflict.



# Termination

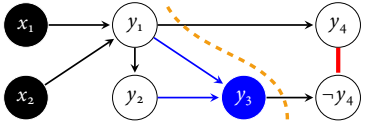
**Important property:**
For the same decisions and universal assignment, adding the conflict clause and removing all clauses right of the cut, would still result in a conflict.




The learnt clause is new. Otherwise, the conflict would have happened earlier.

# Termination

**Important property:**
For the same decisions and universal assignment, adding the conflict clause and removing all clauses right of the cut, would still result in a conflict.




The learnt clause is new. Otherwise, the conflict would have happened earlier.

**Termination:** As there are only exponentially many clauses, Incremental Determinization is guaranteed to terminate.

Incremental Determinization is an **EXPSPACE** algorithm, like CDCL and QCDCL.

# Termination (2)

1: **procedure** IncrementalDeterminization($\forall X.\exists Y.\varphi$)
2: $dlvl \leftarrow 0$; $D \leftarrow \varnothing$
3: **while** true **do**
4: $D, \varphi, conflict, \mathbf{x} \leftarrow$ Propagate($\forall X.\exists Y.\varphi, D, dlvl$)
5: **if** $conflict$ **then**
6: $c \leftarrow$ AnalyzeConflict($\forall X.\exists Y.\ \varphi, \mathbf{x}, D, dlvl$)
7: **if** $c$ only contains variables in $X$ **then**
8: **return** false
9: $dlvl \leftarrow$ (maximal decision level in $c$) $- 1$
10: $\varphi, D \leftarrow$ Backtrack($\varphi, D, dlvl$)
11: $\varphi \leftarrow \varphi \wedge c$
12: **else**
13: **if** $D = Y$ **then**
14: **return** true
15: $v \leftarrow$ PickVar($Y \setminus D$)
16: $dlvl \leftarrow dlvl + 1$
17: $\varphi \leftarrow \varphi \wedge$ Decision($v, \varphi, D$)

# Termination (2)

```
1:  procedure IncrementalDeterminization(∀X.∃Y.φ)
2:      dlvl ← 0;  D ← ∅
3:      while true do
4:          D, φ, conflict, x ← Propagate(∀X.∃Y.φ, D, dlvl)
5:          if conflict then
6:              c ← AnalyzeConflict(∀X.∃Y. φ, x, D, dlvl)
7:              if c only contains variables in X then
8:                  return false
9:              dlvl ← (maximal decision level in c) − 1
10:             φ, D ← Backtrack(φ, D, dlvl)
11:             φ ← φ ∧ c
12:         else
13:             if D = Y then
14:                 return true
15:             v ← PickVar(Y ∖ D)
16:             dlvl ← dlvl + 1
17:             φ ← φ ∧ Decision(v, φ, D)
```

**Invariant:** For every assignment to $X$ the clauses that contain only variables in $D$ admit exactly one assignment to $D$, which can be computed by Boolean constraint propagation.

Termination with *false*: clauses containing only universals are empty ($\equiv$ *false*), when we set the universals to the opposite values.

Termination with *true*: all variables are in $D$. The original clauses, decision clauses, and learnt clauses form a **functional relation**.

# Incremental Determinization (ID) vs CEGAR/QCDCL

- $\forall X \exists Y.\ X = Y$ is solved instantly by ID.
- Empirical performance of ID is good.
- ID does not need preprocessing.

# Incremental Determinization (ID) vs CEGAR/QCDCL

- $\forall X \exists Y.\ X = Y$ is solved instantly by ID.
- Empirical performance of ID is good.
- ID does not need preprocessing.
- ID computes certificates without overhead!

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# Recap: Certification

Given a QBF $\forall x\, \exists y.\ \varphi(x, y)$, a **Skolem function** $f$ is mapping assignments to $x$ to assignments to $y$ such that $\varphi(x, f(x))$ is valid.

A QBF is true if, and only if, it has a Skolem function.

# Recap: Certification

Given a QBF $\forall x\ \exists y.\ \varphi(x, y)$, a **Skolem function** $f$ is mapping assignments to $x$ to assignments to $y$ such that $\varphi(x, f(x))$ is valid.

A QBF is true if, and only if, it has a Skolem function.

Examples

| Formula | Skolem functions |
|---|---|
| $\forall x\ \exists y.\ \mathit{false}$ | - |
| $\forall x\ \exists y.\ \mathit{true}$ | $1, 0, x\%2, x+1, \ldots$ |

# Recap: Certification

Given a QBF $\forall x\ \exists y.\ \varphi(x, y)$, a **Skolem function** $f$ is mapping assignments to $x$ to assignments to $y$ such that $\varphi(x, f(x))$ is valid.

A QBF is true if, and only if, it has a Skolem function.

Examples

| Formula | Skolem functions |
|---|---|
| $\forall x\ \exists y.\ \mathit{false}$ | - |
| $\forall x\ \exists y.\ \mathit{true}$ | 1, 0, $x\%2$, $x + 1$, … |
| $\forall x\ \exists y.\ x\%2$ | - |
| $\forall x\ \exists y.\ y\%2$ | 0,2, $x * 2$, … |
| $\forall x\ \exists y.\ x = y$ | $x$ |

# Recap: Certification

Given a QBF $\forall x\ \exists y.\ \varphi(x, y)$, a **Skolem function** $f$ is mapping assignments to $x$ to assignments to $y$ such that $\varphi(x, f(x))$ is valid.

A QBF is true if, and only if, it has a Skolem function.

Examples

| Formula | Skolem functions |
|---|---|
| $\forall x\ \exists y.\ \mathit{false}$ | - |
| $\forall x\ \exists y.\ \mathit{true}$ | 1, 0, $x\%2$, $x + 1$, … |
| $\forall x\ \exists y.\ x\%2$ | - |
| $\forall x\ \exists y.\ y\%2$ | 0,2, $x * 2$, … |
| $\forall x\ \exists y.\ x = y$ | $x$ |

Let's not consider counter-models (Herbrand functions) in this talk.

# How to Extract Skolem Functions?

QCDCL/CEGAR for 2QBF $\forall X \exists Y.\ \varphi(X, Y)$ with certification:

1. Start with formula $\psi := \mathit{true}$ over variables $X$.
2. Search a solution $x$ for $\psi$.
3. If there is no such assignment, terminate with *true*.
4. Search an assignment $y$ to $\varphi(x, Y)$.
5. If there is no such assignment, terminate with *false*.
6. **Remember $y$.**
7. Exclude all assignments to $X$ for which $\varphi(X, y)$; i.e. $\psi' := \psi \wedge \neg\varphi(X, y)$
8. Go back to step 2.

Certificate:

$$y = \begin{cases} y_1 & \text{if } \varphi(X, y_1) \\ y_2 & \text{if } \varphi(X, y_2) \\ \dots & \\ y_n & \text{if } \varphi(X, y_n) \end{cases}$$

# Certification in Incremental Determinization

ID's invariant guarantees that the final set of clauses forms a **functional relation**. That is, for every $x$ there is exactly one $y$ satisfying all clauses.

But how to turn the clauses into a circuit?

- The variables are ordered (implicitly) by the order they were added to $D$.
- Upon termination all clauses have a unique consequence; implication from "smaller" to "larger" variables.
- Define each variable as the conjunction of its unique consequence clauses in which it occurs positively.

# Certificate extraction in Incremental Determinization - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{\text{unique consequences of } y_1} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{\text{unique consequences of } y_2} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{\text{unique consequences of } y_3} \wedge$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)}_{\text{unique consequences of } y_4}$$

# Certificate extraction in Incremental Determinization - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{\text{unique consequences of } y_1} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{\text{unique consequences of } y_2} \wedge$$

$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{\text{unique consequences of } y_3} \wedge$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_1 \vee \neg y_4)}_{\text{unique consequences of } y_4}$$

$y_1 := x_1 \wedge x_2$ $y_2 := x_2 \vee y_1$ $y_3 := \bigvee \varnothing$ $y_4 := y_1$

# Certificate extraction in Incremental Determinization - Example

$$\forall x_1, x_2.\ \exists y_1, y_2, y_3, y_4.\ \underbrace{(x_1 \vee \neg y_1) \wedge (x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_1)}_{\text{unique consequences of } y_1} \wedge$$

$$\underbrace{(\neg x_2 \vee y_2) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee y_1 \vee \neg y_2)}_{\text{unique consequences of } y_2} \wedge$$

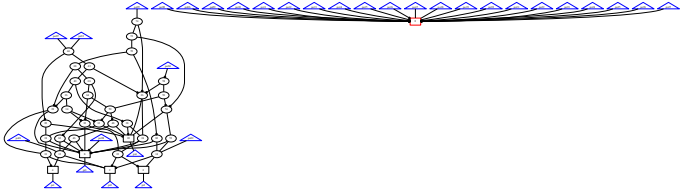
$$\underbrace{(y_1 \vee \neg y_3) \wedge (y_2 \vee \neg y_3) \wedge (\neg y_1 \vee \neg y_3)}_{\text{unique consequences of } y_3} \wedge$$

$$\underbrace{(\neg y_1 \vee y_4) \wedge (\neg y_3 \vee \neg y_4) \wedge (y_3 \vee y_4)}_{\text{unique consequences of } y_4}$$

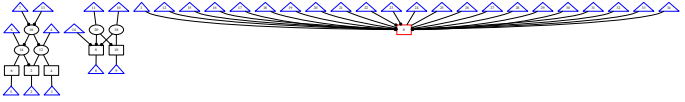
$y_1 := x_1 \wedge x_2$ $y_2 := x_2 \vee y_1$ $y_3 := \bigvee \varnothing$ $y_4 := y_1 \vee \neg y_3$

# ID's certificates can be much smaller - Example

DepQBF



CADET



(stmt6_13_14.qdimacs)

# Logical Problems and their Skolem functions

Consider a quantified Boolean formula (QBF) with universal variables $X = \{x_1, \ldots, x_m\}$ and existential variables $Y = \{y_1, \ldots, y_n\}$.

The formula is true if, and only if, there are Skolem functions $f_1 : X_1 \to \mathbb{B}$, …, $f_n : X_n \to \mathbb{B}$ for the existential variables $y_1$, …, $y_n$. Depending on which logical problem we consider there are additional restrictions on $X_i$:

# Logical Problems and their Skolem functions

Consider a quantified Boolean formula (QBF) with universal variables $X = \{x_1, \ldots, x_m\}$ and existential variables $Y = \{y_1, \ldots, y_n\}$.

The formula is true if, and only if, there are Skolem functions $f_1 : X_1 \to \mathbb{B}$, …, $f_n : X_n \to \mathbb{B}$ for the existential variables $y_1$, …, $y_n$. Depending on which logical problem we consider there are additional restrictions on $X_i$:

QBF with one quantifier alternation: $X = X_1 = \ldots = X_n = X$

Full QBF: $X = X_1 \supseteq \ldots \supseteq X_n$

Dependency QBF: $X_i$ arbitrary

# Logical Problems and their Skolem functions

Consider a quantified Boolean formula (QBF) with universal variables $X = \{x_1, \ldots, x_m\}$ and existential variables $Y = \{y_1, \ldots, y_n\}$.

The formula is true if, and only if, there are Skolem functions $f_1 : X_1 \to \mathbb{B}$, …, $f_n : X_n \to \mathbb{B}$ for the existential variables $y_1$, …, $y_n$. Depending on which logical problem we consider there are additional restrictions on $X_i$:

QBF with one quantifier alternation: $X = X_1 = \ldots = X_n = X$

⇩ Functional Synthesis

Full QBF: $X = X_1 \supseteq \ldots \supseteq X_n$

⇩

Dependency QBF: $X_i$ arbitrary

# (Boolean) Functional Synthesis

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

Example: $(x_1 \rightarrow y) \wedge (x_2 \rightarrow \neg y)$ for $X = \{x_1, x_2\}$ and $Y = \{y\}$

# (Boolean) Functional Synthesis

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

Example: $(x_1 \rightarrow y) \wedge (x_2 \rightarrow \neg y)$ for $X = \{x_1, x_2\}$ and $Y = \{y\}$

Possible truth tables for $f$:

| | |
|---|---|
| $x_1, \neg x_2$ | 1 |
| $\neg x_1, x_2$ | 0 |
| $x_1, x_2$ | $\{0, 1\}$ |
| $\neg x_1, \neg x_2$ | $\{0, 1\}$ |

# How does functional synthesis compare to QBF?

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

Consider this 3QBF: $\psi_3 := \forall X \exists Y.\ \varphi(X, Y) \ \vee \ (\forall Y'.\ \neg\varphi(X, Y'))$

- A Skolem function for $\psi_3$ is a solution to the functional synthesis problem.
- $\psi_3$ is true for any $\varphi$.
- $\Rightarrow$ Exact formulation of functional synthesis.

# How does functional synthesis compare to QBF?

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

Consider this 3QBF: $\psi_3 := \forall X \exists Y.\ \varphi(X, Y) \ \vee \ (\forall Y'.\ \neg\varphi(X, Y'))$

- A Skolem function for $\psi_3$ is a solution to the functional synthesis problem.
- $\psi_3$ is true for any $\varphi$.
- $\Rightarrow$ Exact formulation of functional synthesis.

Compare to this 2QBF: $\psi_2 := \forall X \exists Y.\ \varphi(X, Y)$

- A Skolem function for $\psi_2$ is a solution to the functional synthesis problem.
- $\psi_2$ is not always true.
- **If the 2QBF $\psi_2$ is false, functional synthesis will still compute a function.**

# Functional Synthesis and Projection

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

**Projection problem:**

Given $\varphi(X, Y)$ the projection on $X$ is a formula $\varphi'(X)$ such that $\varphi'(X)$ iff $\exists Y.\ \varphi(X, Y)$ for all $X$.

Projection via functional synthesis:

- Given $\varphi(X, Y)$, compute functional synthesis certificate $f$.
- Projection is $\varphi(X, f(X))$.

# Functional Synthesis and Projection

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

**Projection problem:**

Given $\varphi(X, Y)$ the projection on $X$ is a formula $\varphi'(X)$ such that $\varphi'(X)$ iff $\exists Y.\ \varphi(X, Y)$ for all $X$.

Projection via functional synthesis:

- Given $\varphi(X, Y)$, compute functional synthesis certificate $f$.
- Projection is $\varphi(X, f(X))$.

In particular, we can solve 2QBF through functional synthesis.

# Functional Synthesis and Projection

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

**Projection problem:**

Given $\varphi(X, Y)$ the projection on $X$ is a formula $\varphi'(X)$ such that $\varphi'(X)$ iff $\exists Y.\ \varphi(X, Y)$ for all $X$.

Projection via functional synthesis:

- Given $\varphi(X, Y)$, compute functional synthesis certificate $f$.
- Projection is $\varphi(X, f(X))$.

In particular, we can solve 2QBF through functional synthesis.

$\forall X \exists Y.\ \varphi(X, Y)$ iff the projection of $\varphi(X, Y)$ to $X$ is true.

# Algorithms for Functional Synthesis

**Functional Synthesis**:

Given a Boolean formula $\varphi$ over groups of variables $X$ and $Y$, the functional synthesis problem is to find a function $f$ mapping $X$ to $Y$ such that $\varphi(X, f(X))$ for all $X$ for which there is a $Y$ such that $\varphi(X, Y)$. [Kuncak et al.'11]

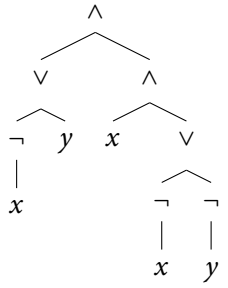
Recent approaches:

- Expansion, BDDs [Fried et al.'16],[Tabaja et al.'17]
- Compositional Functional Synthesis [Akshay et al.'16]
- Incremental Determinization

# Compositional Functional Synthesis [Akshay et al.'17]

Assume we can solve functional synthesis for small formulas, can we build the function for large formulas out of its components?

Consider the formula in negation normal form:




**Build functions bottom-up:** Exploit that function composition for disjunctions is easy. Conjunctions require to apply functional synthesis to the intersection of the two sub-expressions.

# Incremental Determinization for Functional Synthesis

Resolution

$$\frac{C_1 \cup \{l\} \qquad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

Boolean constraint propagation (BCP)

$$\frac{C \cup \{l\} \qquad \{\bar{l}\}}{C}$$

**Observations:**

# Incremental Determinization for Functional Synthesis

Resolution

$$\frac{C_1 \cup \{l\} \qquad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

Boolean constraint propagation (BCP)

$$\frac{C \cup \{l\} \qquad \{\bar{l}\}}{C}$$

**Observations:**

- BCP is a restricted form of resolution.
- BCP is as strong as resolution, if all variables end up having concrete values.

# Incremental Determinization for Functional Synthesis

Resolution

$$\frac{C_1 \cup \{l\} \qquad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

Boolean constraint propagation (BCP)

$$\frac{C \cup \{l\} \qquad \{\bar{l}\}}{C}$$

**Observations:**

- BCP is a restricted form of resolution.
- BCP is as strong as resolution, if all variables end up having concrete values.
- Upon termination, Incremental Determinization computed a set of resolvents (=learnt clauses) and decision clauses with the following property: For all universal assignments, BCP assigns all variables, and the empty clause (=conflict) cannot be derived.

# Incremental Determinization for Functional Synthesis

Resolution

$$\frac{C_1 \cup \{l\} \quad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

Boolean constraint propagation (BCP)

$$\frac{C \cup \{l\} \quad \{\bar{l}\}}{C}$$

**Observations:**

- BCP is a restricted form of resolution.
- BCP is as strong as resolution, if all variables end up having concrete values.
- Upon termination, Incremental Determinization computed a set of resolvents (=learnt clauses) and decision clauses with the following property: For all universal assignments, BCP assigns all variables, and the empty clause (=conflict) cannot be derived.

Different property needed functional synthesis:

For all universal assignments, BCP assigns all variables and **the minimal subsets of clauses needed to derive the empty clause, do not include the decision clauses**.

# Incremental Determinization for Functional Synthesis (cont'd)

Function table view of decisions:

| Condition | $\mathbf{y_4}$ |
|---|---|
| $\neg y_1 \wedge \neg y_2$ | 1 |
| | ?? |
| $y_2$ | 0 |
| $\neg y_3$ | 0 |

Functional: $?? \stackrel{?}{=} \varnothing$

Conflict: $o \cap 1 \stackrel{?}{\neq} \varnothing$

Decision:

assign ?? to 0 or 1

# Incremental Determinization for Functional Synthesis (cont'd)

Function table view of decisions:

| Condition | $y_4$ |
|---|---|
| $\neg y_1 \wedge \neg y_2$ | 1 |
| | ?? |
| $y_2$ | 0 |
| $\neg y_3$ | 0 |

Functional: ?? $\stackrel{?}{=} \varnothing$

Conflict: $o \cap 1 \stackrel{?}{\neq} \varnothing$

Decision:

assign ?? to 0 or 1

Add $\neg(\neg y_1 \wedge \neg y_2) \wedge \neg y_2 \wedge \neg(\neg y_3)$ to the next conflict check, to state that we are only interested in conflicts with this decision.

# Incremental Determinization for Functional Synthesis (cont'd)

Function table view of decisions:

| Condition | $y_4$ |
|---|---|
| $\neg y_1 \wedge \neg y_2$ | 1 |
| | ?? |
| $y_2$ | 0 |
| $\neg y_3$ | 0 |

Functional: ?? $\stackrel{?}{=} \varnothing$

Conflict: $o \cap 1 \stackrel{?}{\neq} \varnothing$

Decision:

assign ?? to 0 or 1

Add $\neg(\neg y_1 \wedge \neg y_2) \wedge \neg y_2 \wedge \neg(\neg y_3)$ to the next conflict check, to state that we are only interested in conflicts with this decision.

In general, we require in the global conflict check that one of the decisions must be used.

# Incremental Determinization for Functional Synthesis

1: **procedure** IncrementalDeterminization($\forall X.\exists Y.\varphi$)
2: $dlvl \leftarrow 0$; $D \leftarrow \varnothing$
3: **while** true **do**
4: $D, \varphi, conflict, \mathbf{x} \leftarrow$ Propagate($\forall X.\exists Y.\varphi, D, dlvl$)
5: **if** $conflict$ **then**
6: $c \leftarrow$ AnalyzeConflict($\forall X.\exists Y.\ \varphi, \mathbf{x}, D, dlvl$)
7: **if** $c$ only contains variables in $X$ **then**
8: **return** false
9: $dlvl \leftarrow$ (maximal decision level in $c$) $- 1$
10: $\varphi, D \leftarrow$ Backtrack($\varphi, D, dlvl$)
11: $\varphi \leftarrow \varphi \wedge c$
12: **else**
13: **if** $D = Y$ **then**
14: **return** true
15: $v \leftarrow$ PickVar($Y \setminus D$)
16: $dlvl \leftarrow dlvl + 1$
17: $\varphi \leftarrow \varphi \wedge$ Decision($v, \varphi, D$)

# Incremental Determinization for Functional Synthesis

1: **procedure** IncrementalDeterminization($\forall X.\exists Y.\varphi$)
2: $dlvl \leftarrow 0$; $D \leftarrow \varnothing$
3: **while** true **do**
4: $D, \varphi, conflict, \mathbf{x} \leftarrow$ Propagate($\forall X.\exists Y.\varphi, D, dlvl$)
5: **if** $conflict$ **then**
6: $c \leftarrow$ AnalyzeConflict($\forall X.\exists Y.\ \varphi, \mathbf{x}, D, dlvl$)
7: **if** $c$ only contains variables in $X$ **then**
8: **return** false
9: $dlvl \leftarrow$ (maximal decision level in $c$) $- 1$
10: $\varphi, D \leftarrow$ Backtrack($\varphi, D, dlvl$)
11: $\varphi \leftarrow \varphi \wedge c$
12: **else**
13: **if** $D = Y$ **then**
14: **return** true
15: $v \leftarrow$ PickVar($Y \setminus D$)
16: $dlvl \leftarrow dlvl + 1$
17: $\varphi \leftarrow \varphi \wedge$ Decision($v, \varphi, D$)

$\Leftarrow$ With the additional constraint learnt clauses must contain existential variables and Incremental Determinization cannot terminate with false anymore.

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# Overview

**QBF with one quantifier alternation**

A Different Take on QBF solving

**2QBF Certification and Functional Synthesis**

From Quantifiers to Functions

**Dependency QBF**

Principled Approaches to Reasoning about Functions

# What is DQBF?

Dependency Quantified Boolean Formulas

**Henkin quantifier**: $\exists y : \{x_1, \ldots, x_n\}. \varphi$

# What is DQBF?

Dependency Quantified Boolean Formulas

**Henkin quantifier**: $\exists y : \{x_1, \ldots, x_n\}.\ \varphi$

- Introduces an *existential* variable $y$;
- Specifies which *universal* variables $y$ may depend on;
- $\{x_1, \ldots, x_n\}$ is *dependency set* of $y$; denoted $dep(y)$.

# What is DQBF?

Dependency Quantified Boolean Formulas

**Henkin quantifier**: $\exists y : \{x_1, \ldots, x_n\}.\ \varphi$

- Introduces an *existential* variable $y$;
- Specifies which *universal* variables $y$ may depend on;
- $\{x_1, \ldots, x_n\}$ is *dependency set* of $y$; denoted $dep(y)$.

Grammar: DQBFs start with a list of Henkin quantifiers, followed by a propositional formula.

# What is DQBF?

Dependency Quantified Boolean Formulas

**Henkin quantifier**: $\exists y : \{x_1, \ldots, x_n\}.\ \varphi$

- Introduces an *existential* variable $y$;
- Specifies which *universal* variables $y$ may depend on;
- $\{x_1, \ldots, x_n\}$ is *dependency set* of $y$; denoted $dep(y)$.

Grammar: DQBFs start with a list of Henkin quantifiers, followed by a propositional formula.

A DQBF is true, if for all existentials $y$ there are functions $f_y : \mathbb{B}^{dep(y)} \to \mathbb{B}$ such that $\varphi[y \,/\, f_y(dep(y))]$ is valid.

# DQBF generalizes QBF

Given a QBF: $\forall x_1.\exists y_1.\forall x_2.\exists y_2\ \ldots\ \forall x_k.\exists y_k.\ \varphi$

# DQBF generalizes QBF

Given a QBF: $\forall x_1.\exists y_1.\forall x_2.\exists y_2\ \ldots\ \forall x_k.\exists y_k.\ \varphi$

Equivalent DQBF: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_1, x_2\}\ \ldots\ \exists y_k : \{x_1, \ldots, x_k\}.\ \varphi$

# DQBF generalizes QBF

Given a QBF: $\forall x_1.\exists y_1.\forall x_2.\exists y_2\ \ldots\ \forall x_k.\exists y_k.\ \varphi$

Equivalent DQBF: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_1, x_2\}\ \ldots\ \exists y_k : \{x_1, \ldots, x_k\}.\ \varphi$

Observations:

- Dependency sets of prenex QBFs are nested.
- The order of Henkin quantifiers does not matter.
- For general DQBF dependency sets may be **incomparable**.

# Functions in DQBF

Consider this example:

$$\exists f : \mathbb{B}^n \to \mathbb{B}.\ \forall X_1, X_2.\ f(X_1) \neq f(X_2)$$

# Functions in DQBF

Consider this example:

$$\exists f : \mathbb{B}^n \to \mathbb{B}.\ \forall X_1, X_2.\ f(X_1) \neq f(X_2)$$

Encode each function application as a Henkin quantifier:

$$\exists f_1 : X_1.\ \exists f_2 : X_2.\ f_1 \neq f_2 \ \wedge \textbf{??}$$

# Functions in DQBF

Consider this example:

$$\exists f : \mathbb{B}^n \to \mathbb{B}.\ \forall X_1, X_2.\ f(X_1) \neq f(X_2)$$

Encode each function application as a Henkin quantifier:

$$\exists f_1 : X_1.\ \exists f_2 : X_2.\ f_1 \neq f_2 \ \wedge \ \underbrace{X_1 = X_2 \to f_1 = f_2}_{\text{functional consistency}}$$

# Functions in DQBF

Consider this example:

$$\exists f : \mathbb{B}^n \to \mathbb{B}.\ \forall X_1, X_2.\ f(X_1) \neq f(X_2)$$

Encode each function application as a Henkin quantifier:

$$\exists f_1 : X_1.\ \exists f_2 : X_2.\ f_1 \neq f_2 \ \wedge \ \underbrace{X_1 = X_2 \to f_1 = f_2}_{\text{functional consistency}}$$

What happens for functions with $k$ function applications?

*Can we state pairwise equality with less than $k^2$ constraints?*

# Functions in DQBF

Consider this example:

$$\exists f : \mathbb{B}^n \to \mathbb{B}.\ \forall X_1, X_2.\ f(X_1) \neq f(X_2)$$

Encode each function application as a Henkin quantifier:

$$\exists f_1 : X_1.\ \exists f_2 : X_2.\ f_1 \neq f_2 \ \wedge \ \underbrace{X_1 = X_2 \to f_1 = f_2}_{\text{functional consistency}}$$

What happens for functions with $k$ function applications?

*Can we state pairwise equality with less than $k^2$ constraints?*

Exploit transitivity of equality: $\bigwedge_{k>1} [X_1 = X_k \to f_1 = f_k]$

# DQBF Examples

Synthesize an adder:

$$\begin{aligned}&\exists add : \mathbb{B}^{32} \times \mathbb{B}^{32} \to \mathbb{B}^{32}.\\ &\quad \forall x \in \mathbb{B}^{32}.\ add(x, 0) = x \ \wedge\\ &\quad \forall x, y \in \mathbb{B}^{32}.\ add(x, y+1) = add(x, y) + 1\end{aligned}$$

Synthesize an invariant:

$$\begin{aligned}&\exists inv : \mathbb{B}^{|s|} \to \mathbb{B}\\ &\quad \forall s.\quad I(s) \to inv(s) \wedge\\ &\quad \forall s.\quad inv(s) \wedge E(s) \to P(s) \wedge\\ &\quad \forall s, s'.\ inv(s) \wedge T(s, s') \to inv(s'),\end{aligned}$$

where $s$ is the program state, $T$ is the transition function of the loop body, $I$ is the initial condition, $E$ is the loop exit condition, and $P$ is the property to prove.

# What are the Principled Approaches to Reasoning About Functions?

How to approach this question at all? **Proof systems.**

# What are the Principled Approaches to Reasoning About Functions?

How to approach this question at all? **Proof systems.**

Proof systems for QBF:

- Resolution+Expansion
- Resolution+$\forall$Red

# What are the Principled Approaches to Reasoning About Functions?

How to approach this question at all? **Proof systems.**

Proof systems for QBF:

| | |
|---|---|
| Resolution+Expansion | BDDs, *quantor* [Biere'04], RAReQS [Janota et al.'11] |
| Resolution+$\forall$Red | DepQBF, Quantor, CAQE, CADET |

# What are the Principled Approaches to Reasoning About Functions?

How to approach this question at all? **Proof systems.**

Proof systems for QBF:

| | |
|---|---|
| Resolution+Expansion | BDDs, *quantor* [Biere'04], RAReQS [Janota et al.'11] |
| Resolution+$\forall$Red | DepQBF, Quantor, CAQE, CADET |

Proof systems for DQBF:

| | |
|---|---|
| Resolution+Expansion/IR-calculus | [Beyersdorff et al.'16] |
| Fork-Resolution (for restricted DQBF) | [R.'17] |

# The Problem: Resolution is incomplete for DQBF

SAT: Build all resolvents; did we derive the empty clause?

QBF: Build all resolvents; did we derive a clause of universals?

DQBF: Not that easy. [Balabanov et al.'14]

# The Problem: Resolution is incomplete for DQBF

SAT: Build all resolvents; did we derive the empty clause?

QBF: Build all resolvents; did we derive a clause of universals?

DQBF: Not that easy. [Balabanov et al.'14]

Example: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.\ (x_1 \wedge x_2) \leftrightarrow (y_1 = y_2)$

In CNF: $(y_1 \vee y_2 \vee x_1) \wedge (\neg y_1 \vee \neg y_2 \vee x_1) \wedge (y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee \neg y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_1 \vee \neg x_2)$

# The Problem: Resolution is incomplete for DQBF

SAT: Build all resolvents; did we derive the empty clause?

QBF: Build all resolvents; did we derive a clause of universals?

DQBF: Not that easy. [Balabanov et al.'14]

Example: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.\ (x_1 \wedge x_2) \leftrightarrow (y_1 = y_2)$

In CNF: $(y_1 \vee y_2 \vee x_1) \wedge (\neg y_1 \vee \neg y_2 \vee x_1) \wedge (y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee \neg y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_1 \vee \neg x_2)$

Formula is false, but resolution cannot derive a new clause.

# The Problem: Resolution is incomplete for DQBF

SAT: Build all resolvents; did we derive the empty clause?

QBF: Build all resolvents; did we derive a clause of universals?

DQBF: Not that easy. [Balabanov et al.'14]

Example: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.\ (x_1 \wedge x_2) \leftrightarrow (y_1 = y_2)$

In CNF: $(y_1 \vee y_2 \vee x_1) \wedge (\neg y_1 \vee \neg y_2 \vee x_1) \wedge (y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee \neg y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_1 \vee \neg x_2)$

Formula is false, but resolution cannot derive a new clause.

There are 1.5 ways to approach this: Expansion and Fork Extension.

# Resolution+Expansion

Given a DQBF $\exists y_1 : X_1. \ \ldots \ \exists y_n : X_n : \ \varphi$, we can **expand** a universal variable $x$ as follows:

$$\exists y_1 : X_1 \setminus \{x\}. \ \ldots \ \exists y_n : X_n \setminus \{x\}. \exists y'_1 : X_1 \setminus \{x\}. \ \ldots \ \exists y'_n : X_n \setminus \{x\} : \\ \varphi[x/0] \wedge \varphi[x/1][y_i/y'_i \text{ for all } i]$$

Step by step:

- Introduce copies $y'_i$ for each existential $y_i$ (with $x \in dep(y_i)$).
- Remove $x$ from all dependency sets.
- Consider two copies of $\varphi$; replace $x$ by 0 in the first, and by 1 in the second.
- In the second copy of $\varphi$, replace $y_i$ by $y'_i$.

# Expansion-based Algorithms

**Resolution+Expansion proof system:** Expand all universal variables, solve the remaining propositional formula using resolution.

# Expansion-based Algorithms

**Resolution+Expansion proof system:** Expand all universal variables, solve the remaining propositional formula using resolution. (Does not work at all in practice.)

# Expansion-based Algorithms

**Resolution+Expansion proof system:** Expand all universal variables, solve the remaining propositional formula using resolution. (Does not work at all in practice.)

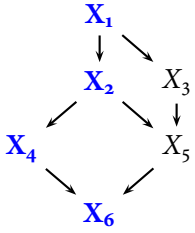
Do we have to expand all universal variables? It suffices to expand until we have a QBF! [Wimmer et al.'17]

# Expansion-based Algorithms

**Resolution+Expansion proof system:** Expand all universal variables, solve the remaining propositional formula using resolution. (Does not work at all in practice.)

Do we have to expand all universal variables? It suffices to expand until we have a QBF! [Wimmer et al.'17]

Given a DQBF, consider the inclusion order of its dependencies:




Still does not really work in practice.

# Towards an Alternative Proof System for DQBF

Example: $\exists y_1 : X_1.\, \exists y_2 : X_2.\, \varphi \wedge (y_1 \vee y_2)$

Who is responsible for satisfying $(y_1 \vee y_2)$?

# Towards an Alternative Proof System for DQBF

Example: $\exists y_1 : X_1.\, \exists y_2 : X_2.\, \varphi \wedge (y_1 \vee y_2)$

Who is responsible for satisfying $(y_1 \vee y_2)$?

SAT ($X_1 = X_2 = \varnothing$): one of them, statically

# Towards an Alternative Proof System for DQBF

Example: $\exists y_1 : X_1.\, \exists y_2 : X_2.\, \varphi \wedge (y_1 \vee y_2)$

Who is responsible for satisfying $(y_1 \vee y_2)$?

- SAT ($X_1 = X_2 = \varnothing$): one of them, statically
- QBF ($X_1 \subseteq X_2$): $y_2$ whenever not satisfied by $y_1$

# Towards an Alternative Proof System for DQBF

Example: $\exists y_1 : X_1.\ \exists y_2 : X_2.\ \varphi \wedge (y_1 \vee y_2)$

Who is responsible for satisfying $(y_1 \vee y_2)$?

- SAT ($X_1 = X_2 = \varnothing$): one of them, statically
- QBF ($X_1 \subseteq X_2$): $y_2$ whenever not satisfied by $y_1$
- DQBF ($X_1 \not\subseteq X_2$ and $X_1 \not\supseteq X_2$): **??**

# Towards an Alternative Proof System for DQBF

Example: $\exists y_1 : X_1.\ \exists y_2 : X_2.\ \varphi \wedge (y_1 \vee y_2)$

Who is responsible for satisfying $(y_1 \vee y_2)$?

- SAT ($X_1 = X_2 = \varnothing$): one of them, statically
- QBF ($X_1 \subseteq X_2$): $y_2$ whenever not satisfied by $y_1$
- DQBF ($X_1 \not\subseteq X_2$ and $X_1 \not\supseteq X_2$): **??**

Clauses containing vars with incomparable dependencies: **Information forks**

Concept loosely connected to distributed reactive synthesis. [Finkbeiner&Schewe'05]

# The Fork-Resolution Proof System [R.'17]

Insight: $y_1$ and $y_2$ must coordinate based on their shared information.

# The Fork-Resolution Proof System [R.'17]

Insight: $y_1$ and $y_2$ must coordinate based on their shared information.

**New proof rule: Fork extension**

$$\frac{C_1 \cup C_2 \qquad dep(C_1) \nsubseteq dep(C_2) \qquad dep(C_1) \not\supseteq dep(C_2) \qquad t \text{ is fresh}}{\exists t : dep(C_1) \cap dep(C_2).\ C_1 \cup \{t\} \ \wedge\ C_2 \cup \{\neg t\}}$$

# The Fork-Resolution Proof System [R.'17]

Insight: $y_1$ and $y_2$ must coordinate based on their shared information.

**New proof rule: Fork extension**

$$\frac{C_1 \cup C_2 \qquad dep(C_1) \nsubseteq dep(C_2) \qquad dep(C_1) \not\supseteq dep(C_2) \qquad t \text{ is fresh}}{\exists t : dep(C_1) \cap dep(C_2).\ C_1 \cup \{t\} \ \wedge \ C_2 \cup \{\neg t\}}$$

Resolution

$$\frac{C_1 \cup \{l\} \qquad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

# The Fork-Resolution Proof System [R.'17]

Insight: $y_1$ and $y_2$ must coordinate based on their shared information.

**New proof rule: Fork extension**

$$\frac{C_1 \cup C_2 \qquad dep(C_1) \nsubseteq dep(C_2) \qquad dep(C_1) \nsupseteq dep(C_2) \qquad t \text{ is fresh}}{\exists t : dep(C_1) \cap dep(C_2).\ C_1 \cup \{t\} \ \wedge \ C_2 \cup \{\neg t\}}$$

Resolution

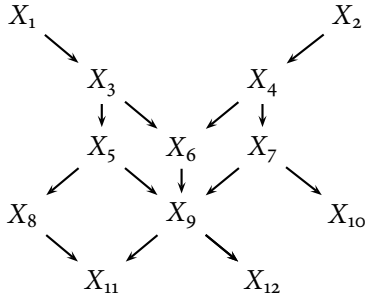
$$\frac{C_1 \cup \{l\} \qquad C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

Universal Reduction

$$\frac{C \quad var(l) \notin dep(C \setminus \{l\}) \quad \bar{l} \notin C \quad var(l) \text{ is universal}}{C \setminus \{l\}}$$

# Fork Resolution

Consider the lattice of dependency sets of a DQBF.




**Algorithm sketch:** Pick a variable from a leaf element of the lattice; eliminate its information forks; eliminate the variable with resolution. Iterate.

# Completeness Guarantee

Fork resolution is complete for DQBF with two disjoint function applications.

For example, consider invariant synthesis:

$$\begin{aligned}&\exists inv : \mathbb{B}^{|s|} \to \mathbb{B}\\ &\quad \forall s. \quad I(s) \to inv(s) \wedge\\ &\quad \forall s. \quad inv(s) \wedge E(s) \to P(s) \wedge\\ &\quad \forall s, s'.\ inv(s) \wedge T(s, s') \to inv(s'),\end{aligned}$$

where $s$ is the program state, $T$ is the transition function of the loop body, $I$ is the initial condition, $E$ is the loop exit condition, and $P$ is the property to prove.

The function to synthesize, $inv$, is only applied to two disjoint arguments.

# Example

Quantifier prefix: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.$

$$y_1 \vee y_2 \vee x_1 \tag{1}$$
$$\neg y_1 \vee \neg y_2 \vee x_1 \tag{2}$$
$$y_1 \vee y_2 \vee x_2 \tag{3}$$
$$\neg y_1 \vee \neg y_2 \vee x_2 \tag{4}$$
$$y_1 \vee \neg y_2 \vee \neg x_1 \vee \neg x_2 \tag{5}$$
$$\neg y_1 \vee y_2 \vee \neg x_1 \vee \neg x_2 \tag{6}$$

All clauses are information forks!

# Example

Quantifier prefix: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.\ \exists t_1 : \varnothing.\ \ldots \exists t_6 : \varnothing.$

| | | | |
|---|---|---|---|
| $t_1 \vee y_1 \vee x_1$ | (1a) | $\neg t_1 \vee y_2$ | (1b) |
| $t_2 \vee \neg y_1 \vee x_1$ | (2a) | $\neg t_2 \vee \neg y_2$ | (2b) |
| $t_3 \vee y_1$ | (3a) | $\neg t_3 \vee y_2 \vee x_2$ | (3b) |
| $t_4 \vee \neg y_1$ | (4a) | $\neg t_4 \vee \neg y_2 \vee x_2$ | (4b) |
| $t_5 \vee y_1 \vee \neg x_1$ | (5a) | $\neg t_5 \vee \neg y_2 \vee \neg x_2$ | (5b) |
| $t_6 \vee \neg y_1 \vee \neg x_1$ | (6a) | $\neg t_6 \vee y_2 \vee \neg x_2$ | (6b) |

Split clauses with fork extension. Fresh variables $t$ are propositional.

# Example

Quantifier prefix: $\exists y_1 : \{x_1\}.\ \exists y_2 : \{x_2\}.\ \exists t_1 : \varnothing.\ \ldots \exists t_6 : \varnothing.$

| | | | |
|---|---|---|---|
| $t_1 \vee t_4$ | $(1a4a)$ | $\neg t_1 \vee \neg t_5$ | $(1b5b)$ |
| $t_3 \vee t_6$ | $(3a6a)$ | $\neg t_2 \vee \neg t_6$ | $(2b6b)$ |
| $t_4 \vee t_5$ | $(4a5a)$ | $\neg t_3 \vee \neg t_4$ | $(3b4b)$ |

Eliminate variables $y_1$ and $y_2$ through variable elimination.

Obtain propositional, unsatisfiable formula.

# Conclusion

- Synthesizing Boolean functions would solve many problems in verification and synthesis.
- Closely connected to certification for quantified Boolean logics.
- Incremental Determinization: an effective algorithm to compute 2QBF certificates.
- Functional synthesis as an extension of 2QBF certification. Incremental Determinization can be modified to solve functional synthesis.
- DQBF can express very general Boolean synthesis problems, but efficient algorithms are still elusive.