Machine Learning for Quantifiers

Mikoláš Janota

ML4SP, 10 August 2025

Czech Technical University

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

MINISTRY OF EDUCATION, YOUTH AND SPORTS

ROBOPROX

Outline

Intro: QBF, Expansion, Games, Careful expansion

Solving QBF

Learning in QBF

Targeting SMT

Towards Synthesizing Terms

Towards Infinite Models

Intro: QBF, Expansion, Games, Careful expansion

SAT and QBF

- SAT — determine if a formula is satisfiable

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y.\ (x \leftrightarrow y)$

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y. \, (x \leftrightarrow y)$
- Quantifications as shorthands for connectives
  ($\forall = \wedge$, $\exists = \vee$)

  Example:

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y.\ (x \leftrightarrow y)$
- Quantifications as shorthands for connectives ($\forall = \wedge$, $\exists = \vee$)

  Example:

  1 $\forall x \exists y.\ (x \leftrightarrow y)$

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y.\ (x \leftrightarrow y)$
- Quantifications as shorthands for connectives ($\forall = \wedge$, $\exists = \vee$)

  Example:

  1. $\forall x \exists y.\ (x \leftrightarrow y)$
  2. $\forall x.\ (x \leftrightarrow 0) \vee (x \leftrightarrow 1)$

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \lor y) \land (x \lor \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y.\ (x \leftrightarrow y)$
- Quantifications as shorthands for connectives ($\forall = \land$, $\exists = \lor$)

  Example:

  1 $\forall x \exists y.\ (x \leftrightarrow y)$
  2 $\forall x.\ (x \leftrightarrow 0) \lor (x \leftrightarrow 1)$
  3 $((0 \leftrightarrow 0) \lor (0 \leftrightarrow 1)) \land ((1 \leftrightarrow 0) \lor (1 \leftrightarrow 1))$

SAT and QBF

- SAT — determine if a formula is satisfiable
- Example: $\{x = 1, y = 0\} \models (x \vee y) \wedge (x \vee \neg y)$
- QBF — for a *Quantified* Boolean formula
- Example: $\forall x \exists y.\ (x \leftrightarrow y)$
- Quantifications as shorthands for connectives ($\forall = \wedge$, $\exists = \vee$)

  Example:

  1. $\forall x \exists y.\ (x \leftrightarrow y)$
  2. $\forall x.\ (x \leftrightarrow 0) \vee (x \leftrightarrow 1)$
  3. $((0 \leftrightarrow 0) \vee (0 \leftrightarrow 1)) \wedge ((1 \leftrightarrow 0) \vee (1 \leftrightarrow 1))$
  4. 1 (True)

Satisfiability Modulo Theories (SMT)

- Example (single instantiations)

$$f : \mathbb{Z} \to \mathbb{Z}$$
$$(\forall x : \mathbb{Z})(f(x) > 0)$$
$$f(0) < 0$$

Satisfiability Modulo Theories (SMT)

- Example (single instantiations)

$$
\begin{aligned}
&f : \mathbb{Z} \to \mathbb{Z} \\
&(\forall x : \mathbb{Z})(f(x) > 0) \\
&f(0) < 0
\end{aligned}
$$

- Example (many instantiations)

$$
\begin{aligned}
&f : \mathbb{Z} \to \mathbb{Z} \\
&(\forall x : \mathbb{Z})(f(x) < f(x+1)) \\
&f(0) > f(100)
\end{aligned}
$$

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.
- $\exists$ wins a game if the matrix becomes true.

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.
- $\exists$ wins a game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.
- $\exists$ wins a game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.
- A QBF is true iff there exists a winning strategy for $\exists$.

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.
- $\exists$ wins a game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.
- A QBF is true iff there exists a winning strategy for $\exists$.

Example

$$(\forall u \exists e)(u \leftrightarrow e)$$

Relation to Two-player Games

- QBF as a two-player game between $\forall$ and $\exists$.
- $\forall$ wins a game if the matrix becomes false.
- $\exists$ wins a game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for $\forall$.
- A QBF is true iff there exists a winning strategy for $\exists$.

Example

$$(\forall u \exists e)(u \leftrightarrow e)$$

$\exists$-player wins by playing $e \triangleq u$.

Solving QBF

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Observe:

- $(\exists x \forall uzw)((u \wedge z \wedge w) \Rightarrow x) \wedge ((\neg u \wedge \neg z \wedge \neg w) \Rightarrow \neg x)$

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Observe:

- $(\exists x \forall uzw)((u \wedge z \wedge w) \Rightarrow x) \wedge ((\neg u \wedge \neg z \wedge \neg w) \Rightarrow \neg x)$
- Expansion by definition: 2^3

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Observe:

- $(\exists x \forall uzw)((u \wedge z \wedge w) \Rightarrow x) \wedge ((\neg u \wedge \neg z \wedge \neg w) \Rightarrow \neg x)$
- Expansion by definition: 2^3
- Sufficient: $u = z = w = 1$ and $u = z = w = 0$

Solving by CEGAR Expansion

$$(\exists \vec{E} \forall \vec{U})(\phi) \equiv (\exists \vec{E}) \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Observe:

- $(\exists x \forall uzw)((u \wedge z \wedge w) \Rightarrow x) \wedge ((\neg u \wedge \neg z \wedge \neg w) \Rightarrow \neg x)$
- Expansion by definition: 2^3
- Sufficient: $u = z = w = 1$ and $u = z = w = 0$
- $(\exists x)(1 \Rightarrow x \wedge 1 \Rightarrow \neg x)$

Solving by CEGAR Expansion

$$(\exists\vec{E}\,\forall\vec{U})(\phi) \equiv (\exists\vec{E}) \bigwedge_{\mu\in 2^{\vec{U}}} \phi[\mu]$$

Solve by SAT $\left(\bigwedge_{\mu\in 2^{\vec{U}}} \phi[\mu]\right)$. Impractical!

Observe:

- $(\exists x \forall uzw)((u \wedge z \wedge w) \Rightarrow x) \wedge ((\neg u \wedge \neg z \wedge \neg w) \Rightarrow \neg x)$
- Expansion by definition: 2^3
- Sufficient: $u = z = w = 1$ and $u = z = w = 0$
- $(\exists x)(1 \Rightarrow x \wedge 1 \Rightarrow \neg x)$
- What is a good expansion?

Solving by CEGAR Expansion Contd.

$$(\exists \vec{E} \forall \vec{U})\ \phi \equiv (\exists \vec{E})\ \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$

Solving by CEGAR Expansion Contd.

$$(\exists \vec{E} \forall \vec{U})\ \phi \equiv (\exists \vec{E})\ \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_0]) = \mu_0$ assignment to $\vec{U}$

Solving by CEGAR Expansion Contd.

$$(\exists \vec{E} \forall \vec{U})\ \phi \equiv (\exists \vec{E})\ \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_0]) = \mu_0$ assignment to $\vec{U}$
- $\texttt{SAT}(\phi[\mu_0]) = \tau_1$ assignment to $\vec{E}$

Solving by CEGAR Expansion Contd.

$$(\exists \vec{E} \forall \vec{U})\ \phi \equiv (\exists \vec{E})\ \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_0]) = \mu_0$ assignment to $\vec{U}$
- $\texttt{SAT}(\phi[\mu_0]) = \tau_1$ assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_1]) = \mu_2$ assignment to $\vec{U}$

Solving by CEGAR Expansion Contd.

$$(\exists \vec{E} \forall \vec{U})\ \phi \equiv (\exists \vec{E})\ \bigwedge_{\mu \in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_0]) = \mu_0$ assignment to $\vec{U}$
- $\texttt{SAT}(\phi[\mu_0]) = \tau_1$ assignment to $\vec{E}$
- $\texttt{SAT}(\neg\phi[\tau_1]) = \mu_2$ assignment to $\vec{U}$
- $\texttt{SAT}(\phi[\mu_0] \wedge \phi[\mu_1]) = \tau_2$ assignment to $\vec{E}$

Solving by CEGAR Expansion Contd.

$$(\exists\vec{E}\,\forall\vec{U})\ \phi \equiv (\exists\vec{E})\ \bigwedge_{\mu\in 2^{\vec{U}}} \phi[\mu]$$

Expand gradually instead: [J. and Marques-Silva, 2011]

- Pick τ_0 arbitrary assignment to $\vec{E}$
- $\mathtt{SAT}(\neg\phi[\tau_0]) = \mu_0$ assignment to $\vec{U}$
- $\mathtt{SAT}(\phi[\mu_0]) = \tau_1$ assignment to $\vec{E}$
- $\mathtt{SAT}(\neg\phi[\tau_1]) = \mu_2$ assignment to $\vec{U}$
- $\mathtt{SAT}(\phi[\mu_0] \wedge \phi[\mu_1]) = \tau_2$ assignment to $\vec{E}$
- After n iterations

$$(\exists\vec{E}\,)\ \bigwedge_{i\in 1..n} \phi[\tau_i]$$

Strengths and Weaknesses

Careful Expansion: Good Example

$$(\exists x \ldots \forall y \ldots)(\phi \wedge y)$$

Setting counter-move $y \triangleq 0$ yields false. Stop.

Careful Expansion: Good Example

$$(\exists x \ldots \forall y \ldots)(\phi \land y)$$

Setting counter-move $y \triangleq 0$ yields false. Stop.

$$(\exists x \ldots \forall y \ldots)(x \lor \phi)$$

Setting candidate $x \triangleq 1$ yields true. Stop.

Careful Expansion: Bad Example

$$(\exists x \forall y)(x \Leftrightarrow y)$$

Careful Expansion: Bad Example

$$(\exists x \forall y)(x \Leftrightarrow y)$$

Necessarily you need to use both:

$\mathtt{SAT}(x \Leftrightarrow 0 \wedge x \Leftrightarrow 1) \ldots$ UNSAT Stop

Careful Expansion: Ugly Example

$$(\exists x_1 x_2 \forall y_1 y_2)(x_1 \Leftrightarrow y_1 \vee x_2 \Leftrightarrow y_2)$$

Careful Expansion: Ugly Example

$$(\exists x_1 x_2 \forall y_1 y_2)(x_1 \Leftrightarrow y_1 \vee x_2 \Leftrightarrow y_2)$$

Necessarily need 2^2 values of y_1, y_2

Learning in QBF

Issue

- CEGAR requires 2^n SAT calls for the formula

$$(\exists x_1 \ldots x_n \forall y_1 \ldots y_n) \left(\bigvee_{i \in 1..n} x_i \Leftrightarrow y_i \right)$$

Issue

- CEGAR requires 2^n SAT calls for the formula

$$(\exists x_1 \ldots x_n \forall y_1 \ldots y_n)\left(\bigvee_{i \in 1..n} x_i \Leftrightarrow y_i\right)$$

- BUT: The formula immediately false if we set $y_i \triangleq \neg x_i$.

$$\left(\exists x_1 \ldots x_n \forall y_1 \ldots y_n. \bigvee_{i \in 1..n} x_i \Leftrightarrow \neg x_i\right) \equiv \left(\exists x_1 \ldots x_n.\ 0\right)$$

Issue

- CEGAR requires 2^n SAT calls for the formula

$$(\exists x_1 \ldots x_n \forall y_1 \ldots y_n) \left(\bigvee_{i \in 1..n} x_i \Leftrightarrow y_i \right)$$

- BUT: The formula immediately false if we set $y_i \triangleq \neg x_i$.

$$\left(\exists x_1 \ldots x_n \forall y_1 \ldots y_n . \bigvee_{i \in 1..n} x_i \Leftrightarrow \neg x_i \right) \equiv \left(\exists x_1 \ldots x_n . \ 0 \right)$$

- Idea: plug in functions rather than constants.

Issue

- CEGAR requires 2^n SAT calls for the formula

$$(\exists x_1 \ldots x_n \forall y_1 \ldots y_n) \left(\bigvee_{i \in 1..n} x_i \Leftrightarrow y_i \right)$$

- BUT: The formula immediately false if we set $y_i \triangleq \neg x_i$.

$$\left(\exists x_1 \ldots x_n \forall y_1 \ldots y_n. \bigvee_{i \in 1..n} x_i \Leftrightarrow \neg x_i\right) \equiv \left(\exists x_1 \ldots x_n. \, 0\right)$$

- Idea: plug in functions rather than constants.
- **Where do we get the functions?**

Use Machine Learning

[J., 2018]

1. Enumerate some candidate-countermove pairs.

Use Machine Learning

[J., 2018]

1. Enumerate some candidate-countermove pairs.
2. Run ML to learn a Boolean function for each variable in the inner quantifier.

Use Machine Learning

[J., 2018]

1. Enumerate some candidate-countermove pairs.
2. Run ML to learn a Boolean function for each variable in the inner quantifier.
3. Strengthen abstraction with the functions.

Use Machine Learning

[J., 2018]

1. Enumerate some candidate-countermove pairs.
2. Run ML to learn a Boolean function for each variable in the inner quantifier.
3. Strengthen abstraction with the functions.
4. Repeat.

Machine Learning Example

x_1	x_2	$\dots$	x_n	y_1	y_2	$\dots$	y_n
0	0	$\dots$	0	1	1	$\dots$	1
1	0	$\dots$	0	0	1	$\dots$	1
0	0	$\dots$	1	1	1	$\dots$	0
0	1	$\dots$	1	1	0	$\dots$	0

Machine Learning Example

x_1	x_2	...	x_n	y_1	y_2	...	y_n
0	0	...	0	1	1	...	1
1	0	...	0	0	1	...	1
0	0	...	1	1	1	...	0
0	1	...	1	1	0	...	0

- After 2 steps: $y_1 \leftarrow \neg x_1$, $y_i \leftarrow 1$ for $i \in 2..n$.

Machine Learning Example

x_1	x_2	$\ldots$	x_n	y_1	y_2	$\ldots$	y_n
0	0	$\ldots$	0	1	1	$\ldots$	1
1	0	$\ldots$	0	0	1	$\ldots$	1
0	0	$\ldots$	1	1	1	$\ldots$	0
0	1	$\ldots$	1	1	0	$\ldots$	0

- After 2 steps: $y_1 \leftarrow \neg x_1$, $y_i \leftarrow 1$ for $i \in 2..n$.
- $SAT(x_1 \Leftrightarrow \neg x_1 \vee \bigvee_{i \in 2..n} x_2 \Leftrightarrow 1)$

Machine Learning Example

x_1	x_2	$\ldots$	x_n	y_1	y_2	$\ldots$	y_n
0	0	$\ldots$	0	1	1	$\ldots$	1
1	0	$\ldots$	0	0	1	$\ldots$	1
0	0	$\ldots$	1	1	1	$\ldots$	0
0	1	$\ldots$	1	1	0	$\ldots$	0

- After 2 steps: $y_1 \leftarrow \neg x_1$, $y_i \leftarrow 1$ for $i \in 2..n$.
- $SAT(x_1 \Leftrightarrow \neg x_1 \vee \bigvee_{i \in 2..n} x_2 \Leftrightarrow 1)$
- After 4 steps: $y_1 \leftarrow \neg x_1 \; y_2 \leftarrow \neg x_2 \ldots$

Machine Learning Example

x_1	x_2	$\ldots$	x_n	y_1	y_2	$\ldots$	y_n
0	0	$\ldots$	0	1	1	$\ldots$	1
1	0	$\ldots$	0	0	1	$\ldots$	1
0	0	$\ldots$	1	1	1	$\ldots$	0
0	1	$\ldots$	1	1	0	$\ldots$	0

- After 2 steps: $y_1 \leftarrow \neg x_1$, $y_i \leftarrow 1$ for $i \in 2..n$.
- $SAT(x_1 \Leftrightarrow \neg x_1 \vee \bigvee_{i \in 2..n} x_2 \Leftrightarrow 1)$
- After 4 steps: $y_1 \leftarrow \neg x_1$ $y_2 \leftarrow \neg x_2$ $\ldots$
- Eventually we learn the right functions.

Implementation Notes (2018)

- Use CEGAR as before.

Implementation Notes (2018)

- Use CEGAR as before.
- Recursion to generalize to multiple levels as before.

Implementation Notes (2018)

- Use CEGAR as before.
- Recursion to generalize to multiple levels as before.
- Refinement as before.

Implementation Notes (2018)

- Use CEGAR as before.
- Recursion to generalize to multiple levels as before.
- Refinement as before.
- Every K refinements, learn new functions from last K samples. Refine with them.

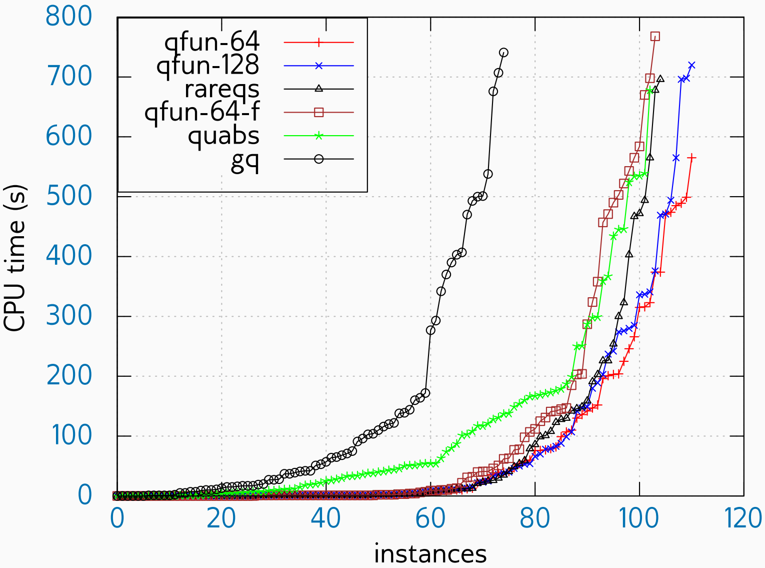
Implementation Notes (2018)

- Use CEGAR as before.
- Recursion to generalize to multiple levels as before.
- Refinement as before.
- Every K refinements, learn new functions from last K samples. Refine with them.
- Learning using decision trees by ID3 algorithm.

Implementation Notes (2018)

- Use CEGAR as before.
- Recursion to generalize to multiple levels as before.
- Refinement as before.
- Every K refinements, learn new functions from last K samples. Refine with them.
- Learning using decision trees by ID3 algorithm.
- Additional heuristic: If a learned function still works, keep it. "Don't fix what ain't broke."

Experiments



Targeting SMT

Herbrand's Theorem

- For FOL $(\forall x \phi)$ is unsatisfiable iff there is unsatisfiable finite grounding with the Herbrand universe

Herbrand's Theorem

- For FOL $(\forall x \phi)$ is unsatisfiable iff there is unsatisfiable finite grounding with the Herbrand universe
- Example

$$\begin{aligned} & f(f(c)) \neq c \\ \wedge \quad & (\forall x)(f(x) = x) \end{aligned}$$

Herbrand's Theorem

- For FOL $(\forall x \phi)$ is unsatisfiable iff there is unsatisfiable finite grounding with the Herbrand universe
- Example

$$
\begin{array}{ll}
 & f(f(c)) \neq c \\
\wedge & (\forall x)(f(x) = x)
\end{array}
$$

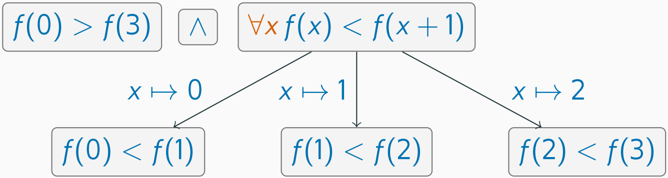
Instantiation:

$$
\begin{array}{ll}
 & f(f(c)) \neq c \\
\wedge & f(c) = c \\
\wedge & f(f(c)) = f(c)
\end{array}
$$

Instantiations for UnSAT

$f(0) > f(3)$ $\wedge$ $\forall x\, f(x) < f(x+1)$

Instantiations for UnSAT



Instantiations for UnSAT

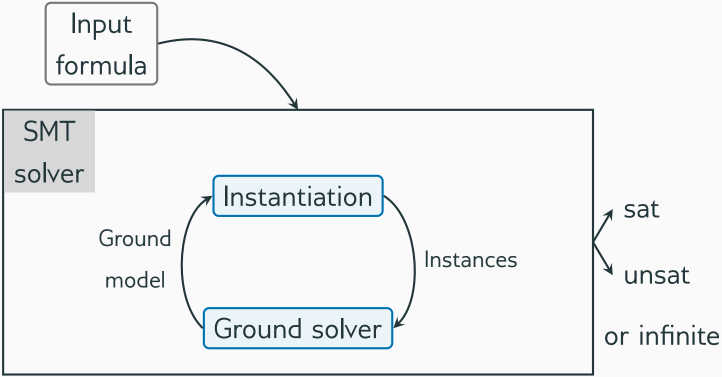
$f(0) > f(3)$ $\wedge$ $\forall x\, f(x) < f(x+1)$

$x \mapsto 0$ $x \mapsto 1$ $x \mapsto 2$

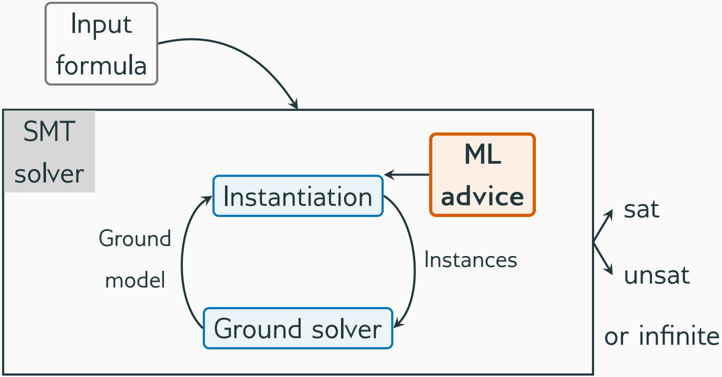
$f(0) < f(1)$ $f(1) < f(2)$ $f(2) < f(3)$

$\Rightarrow \bot$

Setup for Machine Learning



Setup for Machine Learning

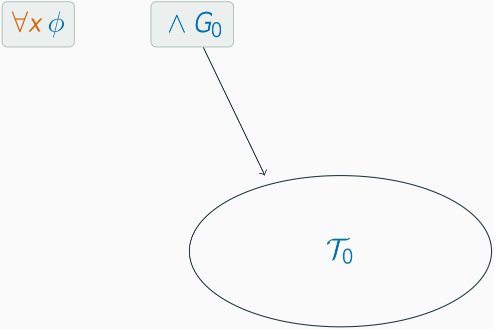


Strengthened Herbrand Theorem

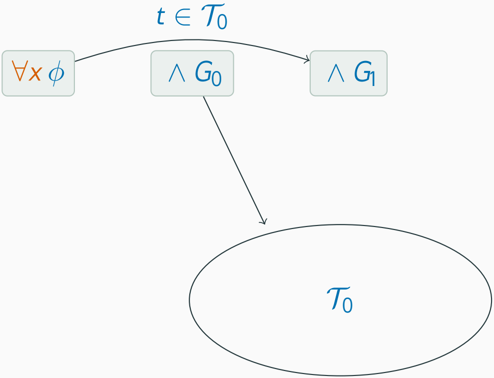
$\forall x \, \phi$

$\wedge \, G_0$

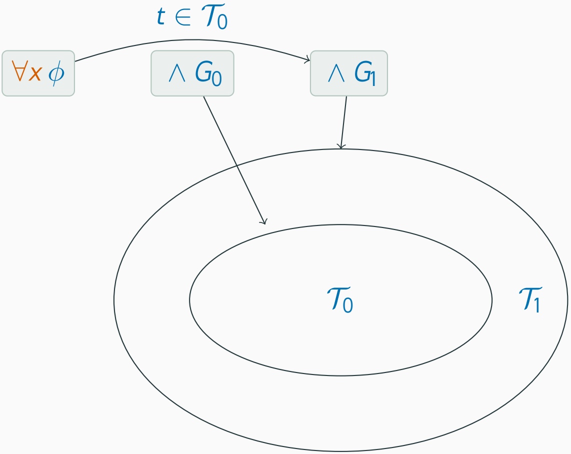
Strengthened Herbrand Theorem



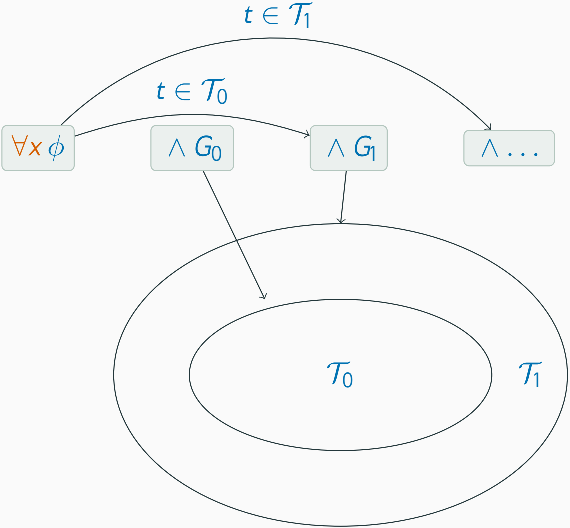
Strengthened Herbrand Theorem



Strengthened Herbrand Theorem



Strengthened Herbrand Theorem



Ordering Possible Candidates

$$\forall \quad x \quad y \quad z \quad (x < y) \vee (x < z)$$

Ordering Possible Candidates

$\forall$	x	y	z
	0	1	11
	3	5	7
	10	9	2

$(x < y) \vee (x < z)$

Ordering Possible Candidates

$\forall$	x	y	z
	0	1	11
	3	5	7
	10	9	2

$$(x < y) \vee (x < z)$$

0	1	11
0	1	7
0	5	11
...	...	...

Ordering Possible Candidates

$\forall$	x	y	z
	10	1	11
	3	5	7
	0	9	2

$(x < y) \vee (x < z)$

Ordering Possible Candidates

∀	x	y	z
	10	1	11
	3	5	7
	0	9	2

$(x < y) \vee (x < z)$

10	1	11
10	1	7
10	5	11
...	...	...

Task for ML

Input:

Set of terms for quantified variable

Task for ML

Input:

Set of terms for quantified variable

Objective:

Order the terms to increase likelihood of UnSAT

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)
- Features are

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)
- Features are
    - **anonymous** for **uninterpreted** symbols, e.g. f, g.

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)
- Features are
  - **anonymous** for **uninterpreted** symbols, e.g. f, g.
  - **non-anonymous** for **interpreted** symbols, e.g. $+, /$.

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)
- Features are
  - **anonymous** for **uninterpreted** symbols, e.g. f, g.
  - **non-anonymous** for **interpreted** symbols, e.g. $+, /$.
- Ground term labelled **positive** if in an existing proof.

High-level ML Design

[J. et al., 2022]

- Gradient boosted trees (LightGBM)
- Features are
  - **anonymous** for **uninterpreted** symbols, e.g. f, g.
  - **non-anonymous** for **interpreted** symbols, e.g. $+, /$.
- Ground term labelled **positive** if in an existing proof.
- Learned forest gives a score to each ground term.

Features

- *bag-of-words* (BOW) features:

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5: *variable, skolem, not, and, plus, forall,* etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) = \{\mathit{forall} : 1, \mathit{variable{:}\ 1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5: *variable, skolem, not, and, plus, forall,* etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) = \{\mathit{forall} : 1, \mathit{variable{:}}\ \mathit{1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) =$
    $\{\mathit{forall} : 1, \mathit{variable{:}\ 1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *age*

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) =$
    $\{\mathit{forall} : 1, \mathit{variable{:}\ 1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *age*
  - *phase*

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) =$
    $\{\mathit{forall} : 1, \mathit{variable}\text{: } 1, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *age*
  - *phase*
  - *depth*

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) =$
    $\{\mathit{forall} : 1, \mathit{variable}{:}\ 1, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *age*
  - *phase*
  - *depth*
  - *tried*

Features

- *bag-of-words* (BOW) features:
    - **kinds** determined by **AST** in cvc5:
      variable, skolem, not, and, plus, forall, etc.
    - count number of occurrences of a symbol
    - for example: $\mathsf{BOW}(\forall x\, 2 + x = \mathsf{skl}_1 + 3) =$
      $\{\mathit{forall} : 1, \mathit{variable: 1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
    - *age*
    - *phase*
    - *depth*
    - *tried*
    - *term context*

Features

- *bag-of-words* (BOW) features:
  - **kinds** determined by **AST** in cvc5:
    variable, skolem, not, and, plus, forall, etc.
  - count number of occurrences of a symbol
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) =$
    $\{\mathit{forall} : 1, \mathit{variable}{:}\ 1, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *age*
  - *phase*
  - *depth*
  - *tried*
  - *term context*
  - *variable context*

Features

- *bag-of-words* (BOW) features:
    - **kinds** determined by **AST** in cvc5:
      variable, skolem, not, and, plus, forall, etc.
    - count number of occurrences of a symbol
    - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) = \{\mathit{forall} : 1, \mathit{variable{:}}\ \mathit{1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
    - *age*
    - *phase*
    - *depth*
    - *tried*
    - *term context*
    - *variable context*
    - *variable frequency*

Obtaining Data

- ML requires large quantities of data

Obtaining Data

- ML requires large quantities of data
- **Where do we get them?**

Obtaining Data

- ML requires large quantities of data
- **Where do we get them?**
    1. Try to solve a set of instances.

Obtaining Data

- ML requires large quantities of data
- **Where do we get them?**
    1. Try to solve a set of instances.
    2. Train on the set of instances solved in Step 1.

Obtaining Data

- ML requires large quantities of data
- **Where do we get them?**
  1. Try to solve a set of instances.
  2. Train on the set of instances solved in Step 1.
  3. Augment solver with learned ML model.

Obtaining Data

- ML requires large quantities of data
- **Where do we get them?**
  1. Try to solve a set of instances.
  2. Train on the set of instances solved in Step 1.
  3. Augment solver with learned ML model.
  4. Go to Step 1.

Evaluation

- Families from SMT lib from **UFNIA**, **UFLIA**

Evaluation

- Families from SMT lib from **UFNIA**, **UFLIA**
- **Holdout** and **Target** sets **75%** / **25%**

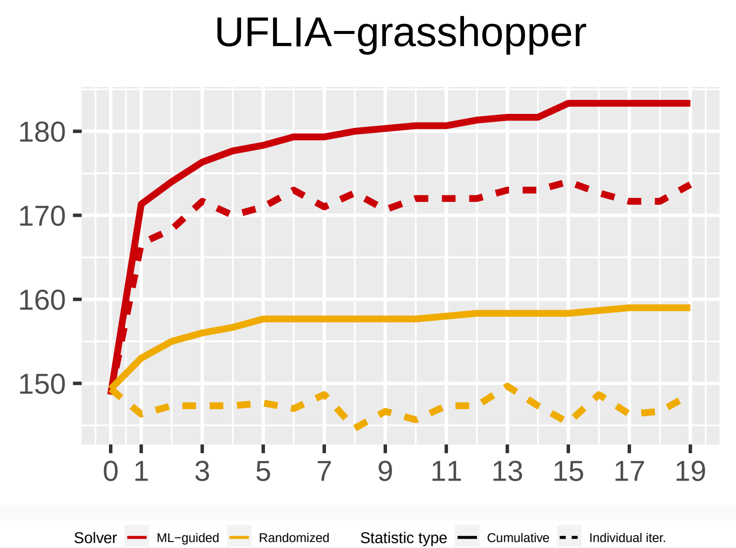
Evaluation

- Families from SMT lib from **UFNIA**, **UFLIA**
- **Holdout** and **Target** sets **75%** / **25%**
- **Cumulative** Goal

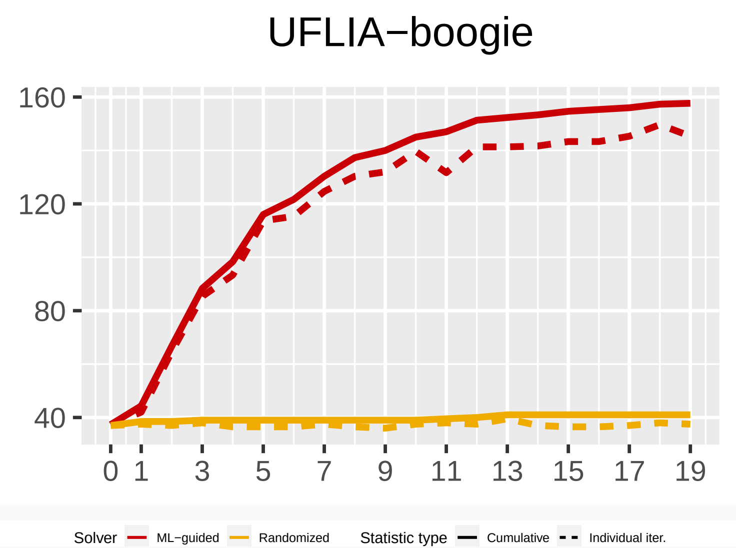
Evaluation

- Families from SMT lib from **UFNIA**, **UFLIA**
- **Holdout** and **Target** sets **75%** / **25%**
- **Cumulative** Goal
- **Single-Instantiation** Goal

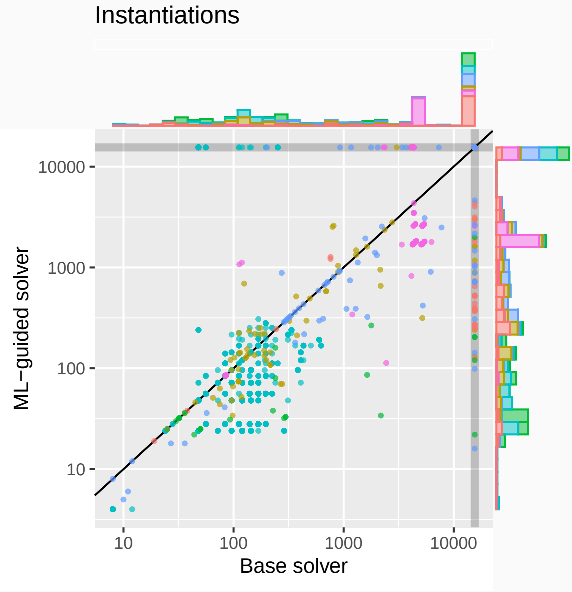
Solved instances – Target set



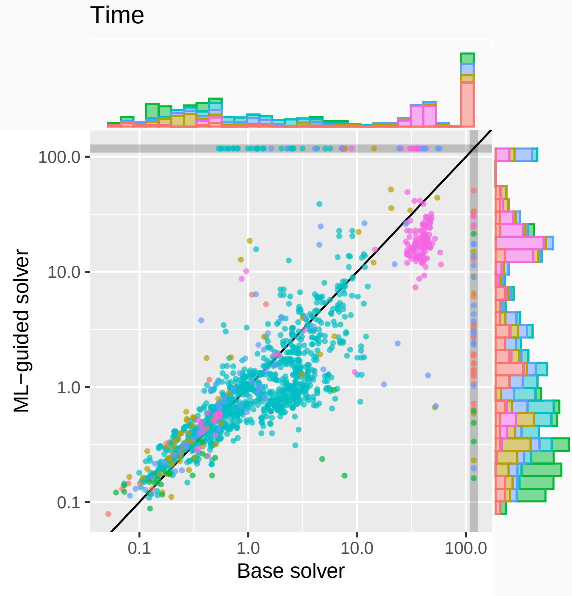
Solved instances – Target set



Holdout Set: Instantiation Count



Holdout Set: Time Comparison

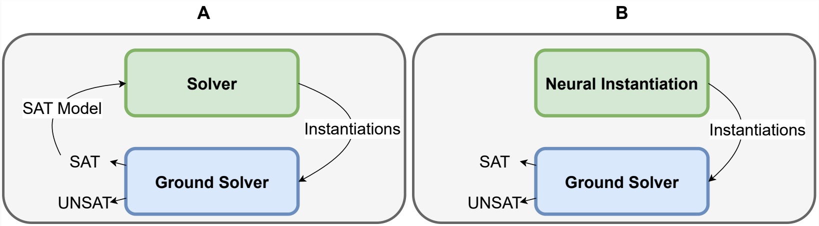


Towards Synthesizing Terms

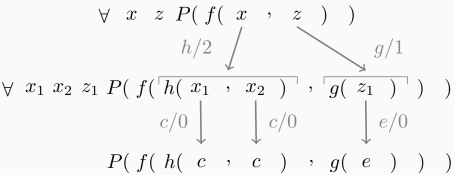
ML Maximalist — Proving By Instantiation

[Piepenbrock et al., 2025]

- A GNN analyzes the formula, and predicts how to instantiate clauses by growing terms
- SAT solver (+ congruence closure) does the rest



Synthesizing Terms



(1) instantiate x by head symbol h with arity 2 and z by g of arity 1 (going from level_0 to level_1)

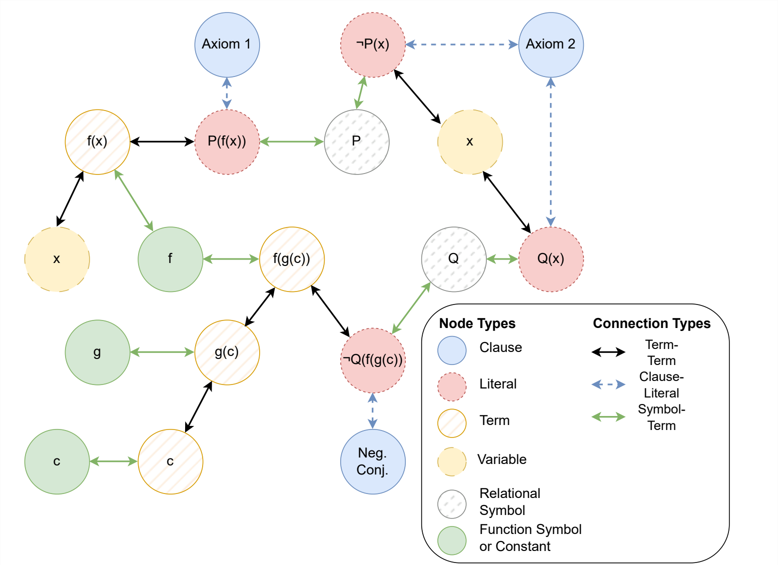
(2) instantiate x_1, x_2, z_1 by constants c, c, and e, respectively (going from level_1 to level_2)

Synthesizing Terms by GNN2RNN

$\forall xz.\, P(f(x,z))$	$x : h \quad z : g$
$\forall x_1 x_2 z_1.\, P(f(h(x_1,x_2),g(z_1)))$	$x_1 : c \xrightarrow{\text{RNN}} x_2 : c \xrightarrow{\text{RNN}} z : e$
$P(f(h(c,c),g(e)))$	

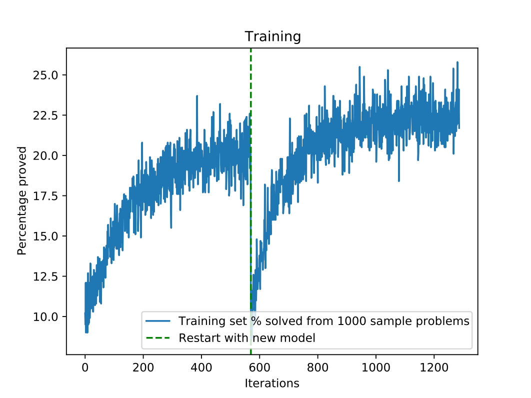
GNN

GNN Example



Axiom 1
¬P(x)
Axiom 2
f(x)
P(f(x))
P
x
x
f
f(g(c))
Q
Q(x)
g
g(c)
¬Q(f(g(c)))
c
c
Neg. Conj.
Node Types
Clause
Literal
Term
Variable
Relational Symbol
Function Symbol or Constant
Connection Types
Term-Term
Clause-Literal
Symbol-Term

System Can Learn



Dedicated Provers are Still Better

Table 1: Performance of various methods. iProver is used in pure instantiation mode. Random is 1 run of the 2-level random grounding. In parentheses, we indicate which dataset was used.

Time limit	1s	10s	60s	Inst. + 30s
Random (all)	—	—	—	3.44%
Neural (train)	—	—	—	26.25%
Neural (test)	—	—	—	19.74%
iProver (train)	43.28%	59.99%	67.6%	—
iProver (test)	43.16%	59.75%	68.69%	—
CVC5 (test)	83.44%	85.6%	86.28%	—

Towards Infinite Models

SMT Models: Constants

```
(declare-fun c () Int)
(declare-fun d () Int)
(assert (< c d))
(check-sat)
(get-model)
```

$c < d$

SMT Models: Constants

```
(declare-fun c () Int)
(declare-fun d () Int)
(assert (< c d))
(check-sat)
(get-model)
```

$c < d$

```
:!z3 ex1.smt2
sat
(
  (define-fun d () Int
    1)
  (define-fun c () Int
    0)
)
```

$c = 0, d = 1$

SMT Models: Functions

```
(declare-fun f (Int) Int)
(assert (< (f 0) (f 1)))
(check-sat)
(get-model)
```

$$f(0) < f(1)$$

SMT Models: Functions

```
(declare-fun f (Int) Int)
(assert (< (f 0) (f 1)))
(check-sat)
(get-model)
```

$f(0) < f(1)$

```
:!z3 ex2.smt2
sat
(
  (define-fun f ((x!0 Int)) Int
    (ite (= x!0 1) 1
      0))
)
```

$fx \triangleq (1 \text{ if } x = 1 \text{ else } 0)$

SMT Models: Quantifiers

```
(declare-fun f (Int) Int)
(assert (forall ((x Int))
           (<= (f x) x)))
(check-sat)
(get-model)
```

$$(\forall x)(fx \leq x)$$

SMT Models: Quantifiers

```
(declare-fun f (Int) Int)
(assert (forall ((x Int))
            (<= (f x) x)))
(check-sat)
(get-model)
```

$(\forall x)(fx \leq x)$

```
:!z3 ex3.smt2
sat
(
  (define-fun f ((x!0 Int)) Int
    x!0)
)
```

$fx \triangleq x$

SMT Models: Quantifiers

```
(declare-fun f (Int) Int)
(assert (forall ((x Int))
             (< (f x) x)))
(check-sat)
(get-model)
```

$$(\forall x)(fx < x)$$

SMT Models: Quantifiers

```
(declare-fun f (Int) Int)
(assert (forall ((x Int))
            (< (f x) x)))
(check-sat)
(get-model)
```

$$(\forall x)(fx < x)$$

```
:!z3 -T:60 ex4.smt2
timeout
```

Not Solved!

Learn infinite models from finite ones?

Model-Based Guided Quantifier Instantiation

For $\forall x \phi$ construct a sequence of:

- candidate models M_i

Model-Based Guided Quantifier Instantiation

For $\forall x \phi$ construct a sequence of:

- candidate models M_i
- counterexample instantiations σ_i

Model-Based Guided Quantifier Instantiation

For $\forall x \phi$ construct a sequence of:

- candidate models M_i
- counterexample instantiations σ_i
- s.t. $M_i \models \bigwedge_{j \in 1..i-1} \phi[x/\sigma_j]$

Model-Based Guided Quantifier Instantiation

For $\forall x \phi$ construct a sequence of:

- candidate models M_i
- counterexample instantiations σ_i
- s.t. $M_i \models \bigwedge_{j \in 1..i-1} \phi[x/\sigma_j]$
- s.t. $M_i \not\models \phi[x/\sigma_i]$

Example

$$(\forall x)(fx > x)$$

$\bigwedge_{j\in 1..i-1}\phi[x/\sigma_j]$	M_i	σ_i
true	$fx \triangleq 0$	$x \mapsto 0$

Example

$$(\forall x)(fx > x)$$

$\bigwedge_{j\in 1..i-1} \phi[x/\sigma_j]$	M_i	σ_i
$f(0) > 0$		

Example

$$(\forall x)(fx > x)$$

$\bigwedge_{j\in 1..i-1} \phi[x/\sigma_j]$	M_i	σ_i
$f(0) > 0$	$fx \triangleq 1$	$x \mapsto 1$

Example

$$(\forall x)(fx > x)$$

$\bigwedge_{j\in 1..i-1} \phi[x/\sigma_j]$	M_i	σ_i
$f(0) > 0$	$fx \triangleq (x = 0\,?\,1 : 2)$	$x \mapsto 2$
$f(1) > 1$		

Example

$$(\forall x)(fx > x)$$

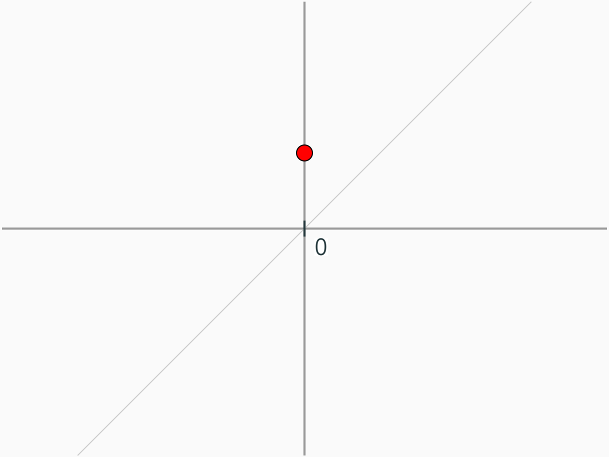
$\bigwedge_{j\in 1..i-1} \phi[x/\sigma_j]$	M_i	σ_i
$f(0) > 0$	$fx \triangleq (x = 0\,?\,1 : (x = 1\,?\,2 : 3))$	
$f(1) > 1$		
$f(2) > 2$		

$$(\forall x)(fx > x)$$

$\bigwedge_{j\in 1..i-1}\phi[x/\sigma_j]$	M_i	σ_i
$f(0) > 0$	$fx \triangleq (x = 0\,?\,1 : (x = 1\,?\,2 : 3))$	
$f(1) > 1$		
$f(2) > 2$		

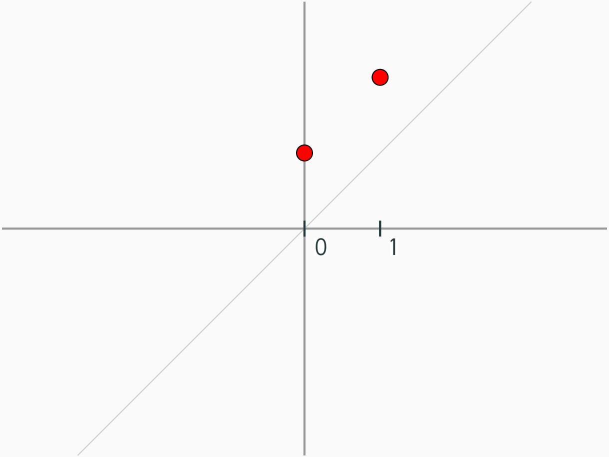
Example: Generalization

$$(\forall x)(fx > x)$$



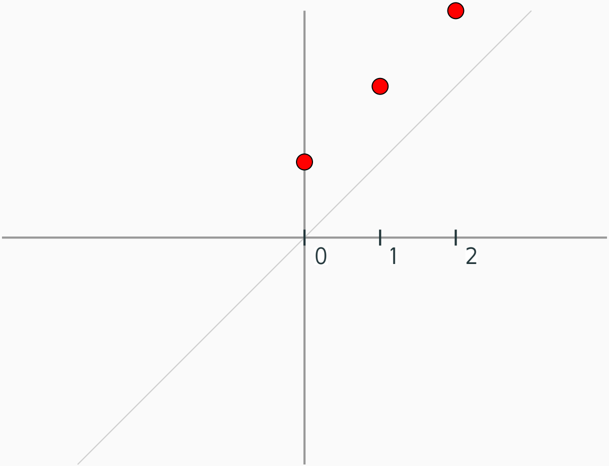
Example: Generalization

$$(\forall x)(fx > x)$$



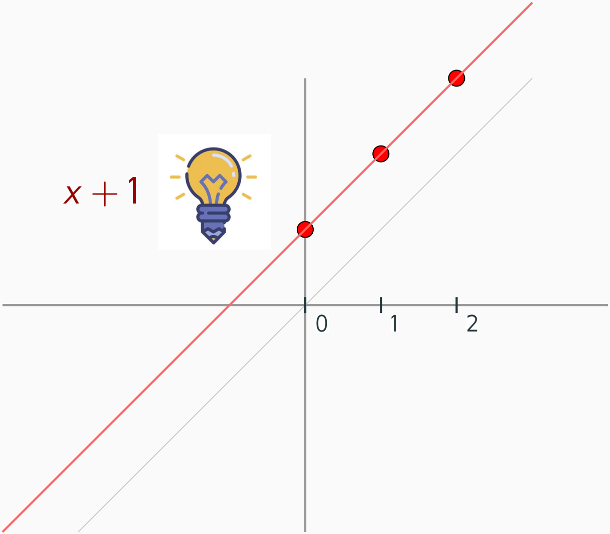
Example: Generalization

$$(\forall x)(fx > x)$$



Example: Generalization

$(\forall x)(fx > x)$



Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.

Generalization for Functions

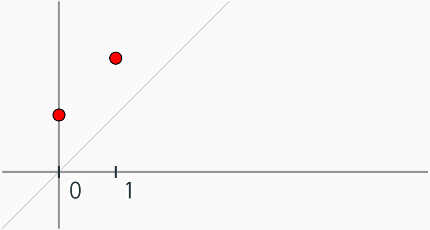
- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**,
  solvable in polynomial time

Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time

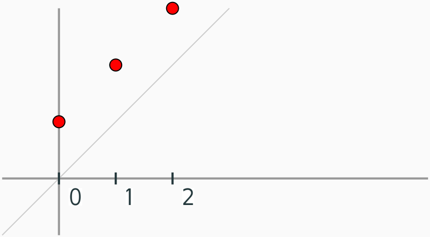
Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time



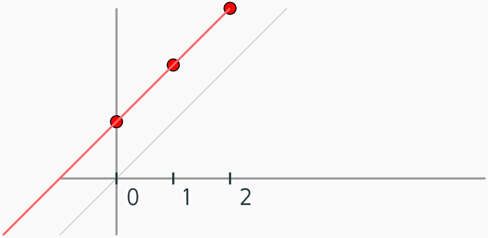
Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time



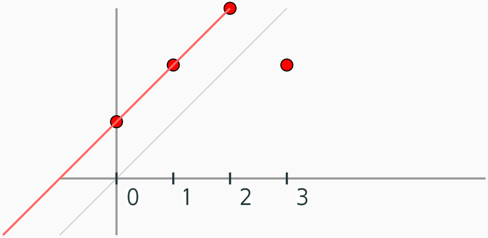
Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**,
  solvable in polynomial time



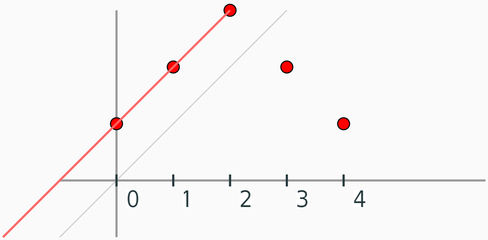
Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time



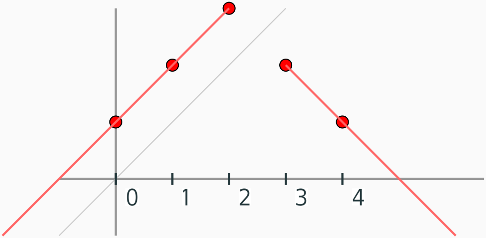
Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time



Generalization for Functions

- Sort points lexicographically
- Keep the same hyper-plane as long as possible
- Otherwise start a new hyper-plane.
- For LIA: linear **Diophantine equations**, solvable in polynomial time

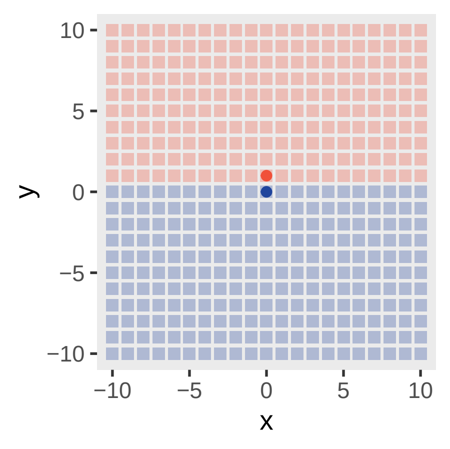


Generalization for Predicates

- Split recursively by hyper-planes
- until all positive or all negative

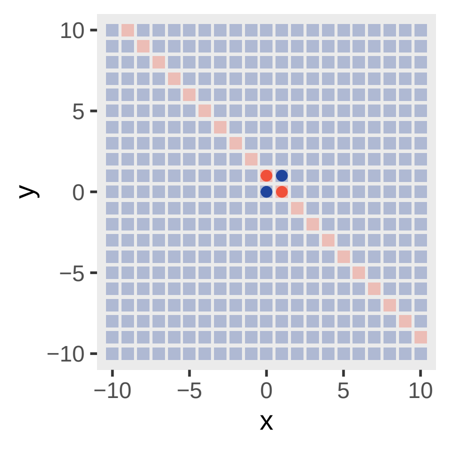
Generalization for Predicates

- Split recursively by hyper-planes
- until all positive or all negative



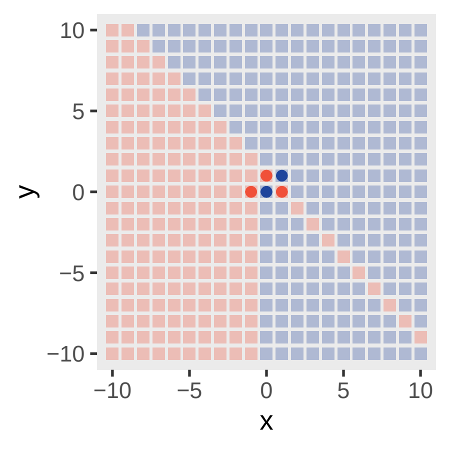
Generalization for Predicates

- Split recursively by hyper-planes
- until all positive or all negative



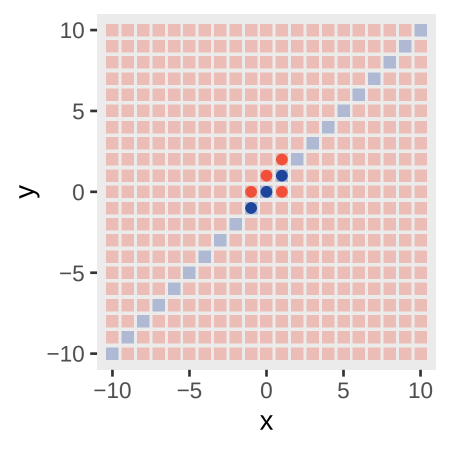
Generalization for Predicates

- Split recursively by hyper-planes
- until all positive or all negative



Generalization for Predicates

- Split recursively by hyper-planes
- until all positive or all negative



Results UFLIA

- Implemented in cvc5
- Run on [J. et al., 2023]

solver	SAT	UNSAT	total
standard MBQI	18,843	7,863	26,706
standard MBQI	18,843	7,863	26,706
ours smart MBQI	31,977	7,863	39,840
Z3	28,380	7,482	35,862

Summary

- Lesson learned from QBF:
  It might be useful to instantiate by more complicated objects, which can be learned.

Summary

- Lesson learned from QBF:
  It might be useful to instantiate by more complicated objects, which can be learned.
- In SMT instantiations can be ordered by ML.

Summary

- Lesson learned from QBF:
  It might be useful to instantiate by more complicated objects, which can be learned.
- In SMT instantiations can be ordered by ML.
- Synthesizing new terms is possible, but harder.

Summary

- Lesson learned from QBF:
  It might be useful to instantiate by more complicated objects, which can be learned.
- In SMT instantiations can be ordered by ML.
- Synthesizing new terms is possible, but harder.
- Synthesizing new models is also possible but
  What are the appropriate ML techniques?

J., M. (2018).
Towards generalization in QBF solving via machine learning.

J., M., Brown, C. E., and Kaliszyk, C. (2023).
A benchmark for infinite models in SMT.

J., M. and Marques-Silva, J. (2011).
Abstraction-based algorithm for 2QBF.

J., M., Piepenbrock, J., and Piotrowski, B. (2022).
Towards learning quantifier instantiation in SMT.

Piepenbrock, J., Urban, J., Korovin, K., Olšák, M., Heskes, T., and J., M. (2025).
Invariant neural architecture for learning term synthesis in instantiation proving.