Theory Solvers for Linear Arithmetic and Equality Logic with Uninterpreted Functions

Martin Köhler

Department of Computer Science
University of Kaiserslautern

2014-11-24

Predicate Logic . . .

. . . is an elegant way to express what we mean:

$$\exists x_{\text{state}}, x'_{\text{state}} : \text{init}(x_{\text{state}}) \wedge \text{bad}(x'_{\text{state}}) \wedge \text{reach}(x_{\text{state}}, x'_{\text{state}})$$

Predicate Logic . . .

. . . is an elegant way to express what we mean:

$$\exists x_{\text{state}}, x'_{\text{state}} : \text{init}(x_{\text{state}}) \wedge \text{bad}(x'_{\text{state}}) \wedge \text{reach}(x_{\text{state}}, x'_{\text{state}})$$

But:

- Satisfiability not decidable

Predicate Logic . . .

. . . is an elegant way to express what we mean:

$$\exists x_{\text{state}}, x'_{\text{state}} : \text{init}(x_{\text{state}}) \wedge \text{bad}(x'_{\text{state}}) \wedge \text{reach}(x_{\text{state}}, x'_{\text{state}})$$

But:

- Satisfiability not decidable
- Some models are non-intuitive: $x < y \wedge y < x$ is satisfiable

Predicate Logic ...

...is an elegant way to express what we mean:

$$\exists x_{\text{state}}, x'_{\text{state}} : \text{init}(x_{\text{state}}) \wedge \text{bad}(x'_{\text{state}}) \wedge \text{reach}(x_{\text{state}}, x'_{\text{state}})$$

But:

- Satisfiability not decidable
- Some models are non-intuitive: $x < y \wedge y < x$ is satisfiable

Idea: Restrict to certain structures

Introducing Theories

Theories restrict our scope to “the interesting” structures

Introducing Theories

Theories restrict our scope to “the interesting” structures

- A theory T is a set of closed formulas that is closed against conclusion

Introducing Theories

Theories restrict our scope to “the interesting” structures

- A theory T is a set of closed formulas that is closed against conclusion
- Here, we treat T as a formula

Introducing Theories

Theories restrict our scope to “the interesting” structures

- A theory T is a set of closed formulas that is closed against conclusion
- Here, we treat T as a formula

Common terms in the context of theories:

Introducing Theories

Theories restrict our scope to "the interesting" structures

- A theory T is a set of closed formulas that is closed against conclusion
- Here, we treat T as a formula

Common terms in the context of theories:

- *T-satisfiable*: At least one relevant structure is a model

Introducing Theories

Theories restrict our scope to “the interesting” structures

- A theory T is a set of closed formulas that is closed against conclusion
- Here, we treat T as a formula

Common terms in the context of theories:

- *T-satisfiable*: At least one relevant structure is a model
- *T-valid*: All relevant structures are models ($\models_T A$)

Introducing Theories

Theories restrict our scope to “the interesting” structures

- A theory T is a set of closed formulas that is closed against conclusion
- Here, we treat T as a formula

Common terms in the context of theories:

- *T-satisfiable*: At least one relevant structure is a model
- *T-valid*: All relevant structures are models ($\models_T A$)
- *T-implication*: Implication restricted to relevant structures ($A \models_T B$)

Satisfiability Modulo Theories (SMT)

Recall: The classical SAT problem is: Given a formula A is there . . .

Satisfiability Modulo Theories (SMT)

Recall: The classical SAT problem is: Given a formula A is there ...

- ... a boolean assignment φ that satisfies the formula?

Satisfiability Modulo Theories (SMT)

Recall: The classical SAT problem is: Given a formula A is there ...

- ... a boolean assignment φ that satisfies the formula?
- ... a predicate logic structure $\mathcal{M}$ that is a model for the formula?

Satisfiability Modulo Theories (SMT)

Recall: The classical SAT problem is: Given a formula A is there ...

- ... a boolean assignment φ that satisfies the formula?
- ... a predicate logic structure $\mathcal{M}$ that is a model for the formula?
  $\uparrow$ undecidable $\uparrow$ maybe a non-standard model

Satisfiability Modulo Theories (SMT)

Recall: The classical SAT problem is: Given a formula A is there ...

- ... a boolean assignment φ that satisfies the formula?
- ... a predicate logic structure $\mathcal{M}$ that is a model for the formula?
  ↑ undecidable ↑ maybe a non-standard model

The Satisfiability Modulo Theory Problem for a theory T: Given a formula A is it satisfied by a model that is allowed by the theory T?

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals:

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \wedge \neg(x < 7)$

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \wedge \neg(x < 7)$
3. Check the conjunct for satisfiability modulo theory (*Theory Solver*)

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \wedge \neg(x < 7)$
3. Check the conjunct for satisfiability modulo theory (*Theory Solver*)
4. Backtrack until either all boolean assignments are checked or a solution in the theory is found

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \land \neg(x < 7)$
3. Check the conjunct for satisfiability modulo theory (*Theory Solver*)
4. Backtrack until either all boolean assignments are checked or a solution in the theory is found

Why does that suffice?

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \wedge \neg(x < 7)$
3. Check the conjunct for satisfiability modulo theory (*Theory Solver*)
4. Backtrack until either all boolean assignments are checked or a solution in the theory is found

Why does that suffice?

- (boolean) unsatisfiable $\rightarrow$ unsatisfiable in theory as well

Theory Solving

1. Boolean SAT methods yield satisfying assignments (boolean).
2. Derive a conjunct of literals: e.g.: $(x < 3) \mapsto \texttt{true}, (x < 7) \mapsto \texttt{false}$ becomes $(x < 3) \land \neg(x < 7)$
3. Check the conjunct for satisfiability modulo theory (*Theory Solver*)
4. Backtrack until either all boolean assignments are checked or a solution in the theory is found

Why does that suffice?

- (boolean) unsatisfiable $\rightarrow$ unsatisfiable in theory as well
- (boolean) satisfiable $\rightarrow$ iff there is a model, a corresponding boolean assignment will be found

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)

- Linear Arithmetic (integer)

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)
- Linear Arithmetic (integer)
- Equality Logic with uninterpreted functions

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)
    - Linear Programming: Simplex Method
- Linear Arithmetic (integer)
- Equality Logic with uninterpreted functions

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)
  - Linear Programming: Simplex Method
- Linear Arithmetic (integer)
  - Integer Linear Programming: Branch-and-Bound
- Equality Logic with uninterpreted functions

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)
  - Linear Programming: Simplex Method
  - Variable elimination: Fourier Motzkin
- Linear Arithmetic (integer)
  - Integer Linear Programming: Branch-and-Bound
- Equality Logic with uninterpreted functions

How to build a Theory Solver?

We have to build Theory Solvers specifically for each Theory.
We will look at:

- Difference Arithmetic (simple example)
- Linear Arithmetic (real)
  - Linear Programming: Simplex Method
  - Variable elimination: Fourier Motzkin
- Linear Arithmetic (integer)
  - Integer Linear Programming: Branch-and-Bound
  - Variable elimination: Omega-Test
- Equality Logic with uninterpreted functions

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.

Example: $x - y \leq 7$

How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.
- Unsatisfiable if and only if the circle has a negative weight

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.
- Unsatisfiable if and only if the circle has a negative weight

How does the graph help?

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.
- Unsatisfiable if and only if the circle has a negative weight

How does the graph help?

- Weights: $y \xrightarrow{7} x$ Reach x from y by walking at most 7

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.
- Unsatisfiable if and only if the circle has a negative weight

How does the graph help?

- Weights: $y \xrightarrow{7} x$ Reach x from y by walking at most 7
- Paths/Walks: Max-lengths of steps imply max-distance of the path

Small example: Difference Arithmetic

Difference Arithmetic: Logic fragment over integers. The predicates define maximal gaps between two variables.
Example: $x - y \leq 7$
How can we solve a conjunction of Difference Arithmetic literals?

- Write variables as nodes in a graph, differences as weights.
- Check for cycles.
- Unsatisfiable if and only if the circle has a negative weight

How does the graph help?

- Weights: $y \xrightarrow{7} x$ Reach x from y by walking at most 7
- Paths/Walks: Max-lengths of steps imply max-distance of the path
- Cycle: Exactly 0 steps from x to x; constraints < 0 cannot be satisfied

Example for Difference Arithmetic

Which of the following conjunctions is satisfiable?

$$A \equiv (y - x \leq 2) \wedge (z - y \leq -3) \wedge (x - z \leq \quad 7)$$
$$B \equiv (v - u \leq 2) \wedge (w - v \leq \quad 3) \wedge (u - w \leq -7)$$

Example for Difference Arithmetic

Which of the following conjunctions is satisfiable?

$$A \equiv (y - x \leq 2) \wedge (z - y \leq -3) \wedge (x - z \leq \ \ 7)$$
$$B \equiv (v - u \leq 2) \wedge (w - v \leq \ \ 3) \wedge (u - w \leq -7)$$

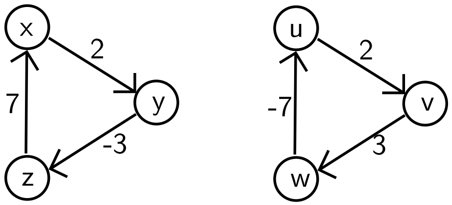
The nodes in the graphs correspond to x, y, z and u, v, w

Example for Difference Arithmetic

Which of the following conjunctions is satisfiable?

$$A \equiv (y - x \leq 2) \wedge (z - y \leq -3) \wedge (x - z \leq \ \ 7)$$
$$B \equiv (v - u \leq 2) \wedge (w - v \leq \ \ 3) \wedge (u - w \leq -7)$$

The nodes in the graphs correspond to x, y, z and u, v, w

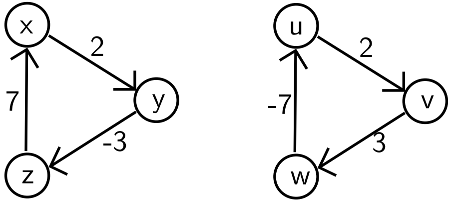


Example for Difference Arithmetic

Which of the following conjunctions is satisfiable?

$$A \equiv (y - x \leq 2) \wedge (z - y \leq -3) \wedge (x - z \leq \ \ 7)$$
$$B \equiv (v - u \leq 2) \wedge (w - v \leq \ \ 3) \wedge (u - w \leq -7)$$

The nodes in the graphs correspond to x, y, z and u, v, w



Left graph: Weight of 6 $\rightarrow$ satisfiable e.g. not $(6, 6, 6)$ but $(7, 3, 0)$
Right graph: Weight of -2 $\rightarrow$ unsat.: u-to-u takes 0 steps but only -7 allowed

The General Simplex Method

We want to allow predicates like $7 \cdot x - 3 \cdot y \leq 42$. *Idea:* Use Simplex Method from linear programming but strip the optimization part. (i.e. General Simplex)

The General Simplex Method

We want to allow predicates like $7 \cdot x - 3 \cdot y \leq 42$. *Idea:* Use Simplex Method from linear programming but strip the optimization part. (i.e. General Simplex)

The **General Simplex Method** requires the input to look like:

- $4 \cdot x + (-7) \cdot y = 0$ (zero check)
- $x \leq 2$ (bounds for variables)

Can we bring any literal in that form?

The General Simplex Method

We want to allow predicates like $7 \cdot x - 3 \cdot y \leq 42$. *Idea:* Use Simplex Method from linear programming but strip the optimization part. (i.e. General Simplex)

The **General Simplex Method** requires the input to look like:

- $4 \cdot x + (-7) \cdot y = 0$ (zero check)
- $x \leq 2$ (bounds for variables)

Can we bring any literal in that form? Indeed, we can!

Example:

- $x - y - 1 \leq 1$

The General Simplex Method

We want to allow predicates like $7 \cdot x - 3 \cdot y \leq 42$. *Idea:* Use Simplex Method from linear programming but strip the optimization part. (i.e. General Simplex)

The **General Simplex Method** requires the input to look like:

- $4 \cdot x + (-7) \cdot y = 0$ (zero check)
- $x \leq 2$ (bounds for variables)

Can we bring any literal in that form? Indeed, we can!

Example:

- $x - y - 1 \leq 1$
- $x - y \leq 2$ (isolate constants)

The General Simplex Method

We want to allow predicates like $7 \cdot x - 3 \cdot y \leq 42$. *Idea:* Use Simplex Method from linear programming but strip the optimization part. (i.e. General Simplex)

The **General Simplex Method** requires the input to look like:

- $4 \cdot x + (-7) \cdot y = 0$ (zero check)
- $x \leq 2$ (bounds for variables)

Can we bring any literal in that form? Indeed, we can!

Example:

- $x - y - 1 \leq 1$
- $x - y \leq 2$ (isolate constants)
- $x - y - s = 0$ (s is a fresh variable) $s \leq 2$

Running the General Simplex

Basic idea: Adjust variables until they fit.

- Make sure the zero-checks are satisfied (How?)

Running the General Simplex

Basic idea: Adjust variables until they fit.

- Make sure the zero-checks are satisfied (How?)
  $\rightarrow$ Initialize all variables with 0
- Goal: Adjust assignments to meet bounds

Running the General Simplex

Basic idea: Adjust variables until they fit.

- Make sure the zero-checks are satisfied (How?)
  $\rightarrow$ Initialize all variables with 0
- Goal: Adjust assignments to meet bounds
- How? Swapping variables with and without bounds (pivoting)

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. (Why?)

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable
- Identify upper and lower bounds

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable
- Identify upper and lower bounds
  - Only one kind of bound: "unbounded" $\rightarrow$ ignore inequalities with this variable (Why?)

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable
- Identify upper and lower bounds
  - Only one kind of bound: “unbounded” $\rightarrow$ ignore inequalities with this variable
  - Both bounds: “bounded” $\rightarrow$ derive implicit inequalities? (What?)

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable
- Identify upper and lower bounds
  - Only one kind of bound: "unbounded" $\rightarrow$ ignore inequalities with this variable
  - Both bounds: "bounded" $\rightarrow$ derive implicit inequalities?

Example: $7 \cdot y - 3 \cdot z \leq x; x \leq -2 \cdot y + 5z$ Ignore x but keep "gaps" for it:
$7 \cdot y - 3 \cdot z \leq -2 \cdot y + 5z$

Variable Elimination: Fourier-Motzkin

Assumption: We only have inequalities. Solve equalities for a variable, substitute everywhere.

Basic idea: Iteratively eliminate variables:

- Solve inequalities for a variable
- Identify upper and lower bounds
  - Only one kind of bound: “unbounded” $\rightarrow$ ignore inequalities with this variable
  - Both bounds: “bounded” $\rightarrow$ derive implicit inequalities?

Trivial once only one variable left

Example: $7 \cdot y - 3 \cdot z \leq x; x \leq -2 \cdot y + 5z$ Ignore x but keep “gaps” for it: $7 \cdot y - 3 \cdot z \leq -2 \cdot y + 5z$

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

- Ask Simplex for a solution.
  - No solution:
  - Integer solution: problem solved.
  - Non-integer solution:

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

- Ask Simplex for a solution.
  - No solution: terminate recursive call
  - Integer solution: problem solved.
  - Non-integer solution:

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

- Ask Simplex for a solution.
  - No solution: terminate recursive call
  - Integer solution: problem solved.
  - Non-integer solution: introduce bounds, recursive calls

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

- Ask Simplex for a solution.
  - No solution: terminate recursive call
  - Integer solution: problem solved.
  - Non-integer solution: introduce bounds, recursive calls
    Example: Solution is $x = 7.0, y = 6.9$, two recursive calls: one with $y \leq 6$ and one with $7 \leq y$

Integer Linear Programming: Branch and Bound

Idea: Use Simplex method to find solutions, try to restrict to integer solutions.

- Ask Simplex for a solution.
  - No solution: terminate recursive call
  - Integer solution: problem solved.
  - Non-integer solution: introduce bounds, recursive calls
    Example: Solution is $x = 7.0, y = 6.9$, two recursive calls: one with $y \leq 6$ and one with $7 \leq y$
- No recursive call finds a solution? Unsatisfiable.

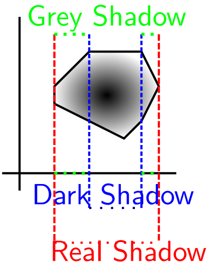
Variable Elimination: Omega Test

Idea: Recursive procedure. Eliminate variable, try three different types of additional constraints recursively.

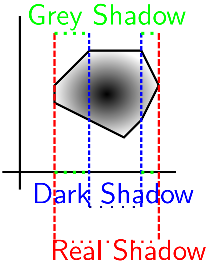
Variable Elimination: Omega Test

Idea: Recursive procedure. Eliminate variable, try three different types of additional constraints recursively.

Adding constraints: The constraints describe where we search (recursively)

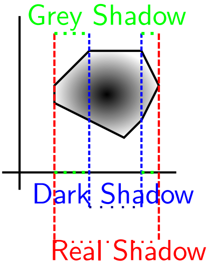


Variable Elimination: Omega Test



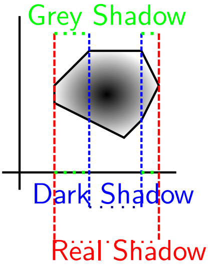
- **Real Shadow:** Eliminate variable as before

Variable Elimination: Omega Test



- **Dark Shadow:** Constraints enforce "big" gaps for the eliminated variable

Variable Elimination: Omega Test



- **Grey Shadow:** Real Shadow without dark shadow

How can we search in the shadow?

Assume we eliminated the variable y.

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)
  - *Solution:* If y is unbounded, a suitable solution can be found.
  - *Solution:* If y is bounded, the solution might not work with an integer y

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)
  - *Solution:* If y is unbounded, a suitable solution can be found.
  - *Solution:* If y is bounded, the solution might not work with an integer y
- **Dark shadow:** Under-approximation, since we only search in the wide parts

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)
  - *Solution:* If y is unbounded, a suitable solution can be found.
  - *Solution:* If y is bounded, the solution might not work with an integer y
- **Dark shadow:** Under-approximation, since we only search in the wide parts
  - *Solution:* We successfully found an integer solution

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)
  - *Solution:* If y is unbounded, a suitable solution can be found.
  - *Solution:* If y is bounded, the solution might not work with an integer y
- **Dark shadow:** Under-approximation, since we only search in the wide parts
  - *Solution:* We successfully found an integer solution
  - *No solution:* Even in a narrow gap might be an integer solution for y.

How can we search in the shadow?

Assume we eliminated the variable y.

- **Real shadow**: Over-approximation, since we ignore y
  - *No solution:* no solution at all (adding y won't yield integer solutions)
  - *Solution:* If y is unbounded, a suitable solution can be found.
  - *Solution:* If y is bounded, the solution might not work with an integer y
- **Dark shadow:** Under-approximation, since we only search in the wide parts
  - *Solution:* We successfully found an integer solution
  - *No solution:* Even in a narrow gap might be an integer solution for y.
- **Grey Shadow:** Excluding the dark shadow from the grey shadow yields a finite set of possible constraints. Try them all.

Equality Logic

Only predicate: Equality of two variables.
Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$

Equality Logic

Only predicate: Equality of two variables.

Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$

Draw variables as nodes of a graph

Equality Logic

Only predicate: Equality of two variables.

Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$

Draw edges for equal and not-equal

Equality Logic

Only predicate: Equality of two variables.

Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$

Transitive closure for equal-edges

Equality Logic

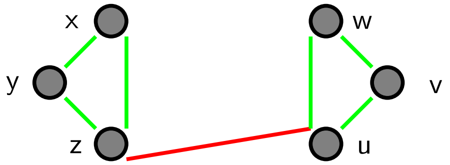
Only predicate: Equality of two variables.

Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$

Connected components get the same value

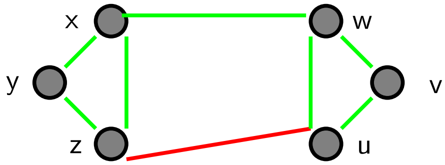
Equality Logic

Only predicate: Equality of two variables.
Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w)$
What if we add $(x = w)$?



Equality Logic

Only predicate: Equality of two variables.
Example: $(x = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = w) \ \wedge (x = u)$
What if we add $(x = w)$?



Equality Logic with Uninterpreted Functions

We want to add functions.
First try: Introduce variables for each term:

$$f(g(y_1), g(y_2)) \Rightarrow x_1 \mapsto g(y_1), x_2 \mapsto g(y_2), x_3 \mapsto f(g(y_1), g(y_2))$$

Equality Logic with Uninterpreted Functions

We want to add functions.
First try: Introduce variables for each term:

$$f(g(y_1), g(y_2)) \Rightarrow x_1 \mapsto g(y_1), x_2 \mapsto g(y_2), x_3 \mapsto f(g(y_1), g(y_2))$$

Ensuring functional consistency: $(y_1 = y_2) \rightarrow (x_1 = x_2)$

Equality Logic with Uninterpreted Functions

We want to add functions.
First try: Introduce variables for each term:

$$f(g(y_1), g(y_2)) \Rightarrow x_1 \mapsto g(y_1), x_2 \mapsto g(y_2), x_3 \mapsto f(g(y_1), g(y_2))$$

Ensuring functional consistency: $(y_1 = y_2) \rightarrow (x_1 = x_2)$
Bugs:

Equality Logic with Uninterpreted Functions

We want to add functions.
First try: Introduce variables for each term:

$$f(g(y_1), g(y_2)) \Rightarrow x_1 \mapsto g(y_1), x_2 \mapsto g(y_2), x_3 \mapsto f(g(y_1), g(y_2))$$

Ensuring functional consistency: $(y_1 = y_2) \rightarrow (x_1 = x_2)$
Bugs:

- Formulas become huge

Equality Logic with Uninterpreted Functions

We want to add functions.
First try: Introduce variables for each term:

$$f(g(y_1), g(y_2)) \Rightarrow x_1 \mapsto g(y_1), x_2 \mapsto g(y_2), x_3 \mapsto f(g(y_1), g(y_2))$$

Ensuring functional consistency: $(y_1 = y_2) \rightarrow (x_1 = x_2)$
Bugs:

- Formulas become huge
- Implications are not conjunctions

Congruence Closure Algorithm

Example:

$$(f(z) = y) \land (y = z) \land \neg(z = u) \land (u = v) \land (v = f(y))$$

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$

Congruence Closure Algorithm

Example:

$$(f(z) = y) \land (y = z) \land \neg(z = u) \land (u = v) \land (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:
  $\{f(z), y, z\}$ $\{u, v, f(y)\}$

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:
  $\{f(z), y, z\}$ $\{u, v, f(y)\}$
- Same function with parameters that are already in the same set: Unite.

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:
  $\{f(z), y, z\}$ $\{u, v, f(y)\}$
- Same function with parameters that are already in the same set: Unite.
  $\{f(z), y, z, u, v, f(y)\}$

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:
  $\{f(z), y, z\}$ $\{u, v, f(y)\}$
- Same function with parameters that are already in the same set: Unite.
  $\{f(z), y, z, u, v, f(y)\}$
- Check for unequal terms in the same set

Congruence Closure Algorithm

Example:

$$(f(z) = y) \wedge (y = z) \wedge \neg(z = u) \wedge (u = v) \wedge (v = f(y))$$

- Put equal terms in the same set:
  $\{f(z), y\}$ $\{y, z\}$ $\{u, v\}$ $\{v, f(y)\}$
- Unite all sets that share at least one term:
  $\{f(z), y, z\}$ $\{u, v, f(y)\}$
- Same function with parameters that are already in the same set: Unite.
  $\{f(z), y, z, u, v, f(y)\}$
- Check for unequal terms in the same set
  $\{f(z), y, z, u, v, f(y)\}$

Thank you for your Attention