Fast Approximations of Quantifier Elimination

Isabel Garcia-Contreras* Hari Govind V K*

Sharon Shoham Arie Gurfinkel

@CAV 2023

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

UNIVERSITY OF WATERLOO

* equal contribution

What is existential quantifier elimination (qelim)?

Given a formula $\varphi \triangleq \exists \boldsymbol{v} \cdot A(\boldsymbol{v})$, find a quantifier-free ψ that is equivalent to φ.

original	qelim
$\exists x \cdot f(x) > 5 \land x \approx y$	$f(y) > 5$
$\exists x \cdot x > 5 \land y > x$	$y > 6$
$\exists a \cdot a[i] \approx w \land a[j] \approx x \land a[k] \approx y \land a[l] \approx z$	$(i \approx j \rightarrow w \approx x) \land (i \approx k \rightarrow w \approx y) \land$ $(i \approx l \rightarrow w \approx z) \land (j \approx k \rightarrow x \approx y) \land$ $(j \approx l \rightarrow y \approx z) \land (i \approx j \rightarrow y \approx z)$
$\exists x \cdot f(x) > 5$	Does not exist

Why qelim?

Widely used in automated reasoning tasks

Playing with Quantified Satisfaction

Nikolaj Bjørner[1] and Mikoláš Janota[2]

[1] Microsoft Research, Redmond, USA
[2] Microsoft Research, Cambridge, UK

Abstract

We develop an algorithm for satisfiability of quantified formulas. The algorithm is based on recent progress in solving Quantified Boolean Formulas, but it generalizes beyond propositional logic to theories, such as linear arithmetic over integers (Presburger arithmetic), linear arithmetic over reals, algebraic data-types and arrays. Compared with previous algorithms for satisfiability of quantified arithmetical formulas our new implementation outperforms previous implementations in Z3 by a significant margin.

Solving Exists/Forall Problems With Yices

Extended Abstract

Bruno Dutertre

Computer Science Laboratory
SRI International
Bruno.Dutertre@sri.com

Abstract

Yices now includes a solver for Exists/Forall problem. We describe the problem, a general solving algorithm, and a key model-based generalization procedure. We explain the Yices implementation of these algorithms and survey a few applications.

1 Introduction

The traditional SMT problem is to determine whether a quantifier-free formula $\Phi(x)$ is satisfiable. Some solvers can also handle first-order formulas with arbitrary quantifiers. We are concerned with a simpler case, namely, formulas of the form $\forall y.\Phi(x, y)$, where $\Phi(x, y)$ is quantifier-free. Implicitly, the variables x are existentially quantified: we are checking the

Complete Functional Synthesis

Viktor Kuncak Mikaël Mayer Ruzica Piskac Philippe Suter *

School of Computer and Communication Sciences (I&C) - Swiss Federal Institute of Technology (EPFL), Switzerland
firstname.lastname@epfl.ch

Abstract

Synthesis of program fragments from specifications can make programs easier to write and easier to reason about. To integrate synthesis into programming languages, synthesis algorithms should behave in a predictable way—they should succeed for a

requires detailed specifications, which for large programs become difficult to write.

We therefore expect that practical applications of synthesis lie in its integration into the compilers of general-purpose programming languages. To make this integration feasible, we aim to identify

SMT-Based Model Checking for Recursive Programs

Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki

Carnegie Mellon University, Pittsburgh, PA, USA

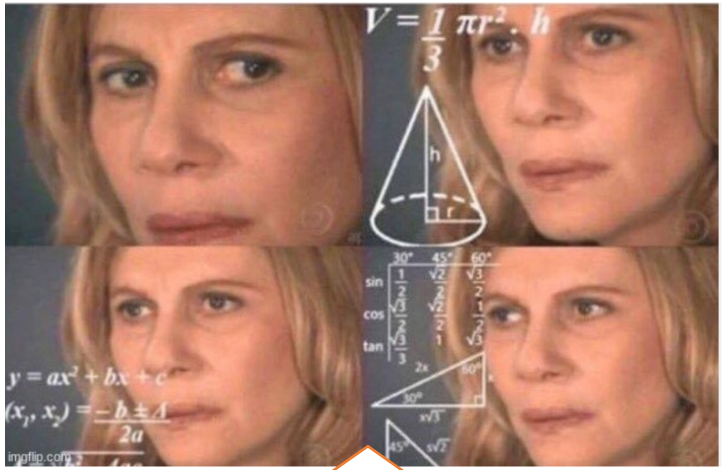
Abstract. We present an SMT-based symbolic model checking algorithm for safety verification of recursive programs. The algorithm is modular and analyzes procedures individually. Unlike other SMT-based approaches, it maintains both *over-* and *under-approximations* of procedure summaries. Under-approximations are used to analyze procedure calls without inlining. Over-approximations are used to block infeasi-

Dealing with qelim

Expensive in general

propositional: PSPACE-complete

LIA: $O(m^{2^n})$



Btw, they should replace the math in this image by qelim problems…

… but cheap in some cases

Existing solvers try their luck with a light preprocess based on the variable substitution rule:

$\exists x \cdot x \approx t \wedge \varphi \equiv \varphi[x \mapsto t]$ (*) provided t is x-free

Dealing with qelim by substitution

$\exists x \cdot x \approx t \land \varphi \equiv \varphi[x \mapsto t]$

Let's try: $\exists x, y \cdot A(x, y)$ with

$$A(x, y) \triangleq y \approx f(x) \land x \approx g(y) \land f(x) \approx 6$$

Trial #1: $A[y \to f(x)]$: $x \approx g(f(x)) \land f(x) \approx 6$ — No more defs

Trial #2: $A[x \to g(y)]$: $y \approx f(g(y)) \land f(g(y)) \approx 6$ — No more defs

Trial #3: $A[y \to 6][x \to g(6)]$: $6 \approx f(g(6))$ — qelim!

by transitivity

Relies on definitions **syntactically existing** in the formula
Depends on **substitution order**
Difficult to deal with **circular equalities**

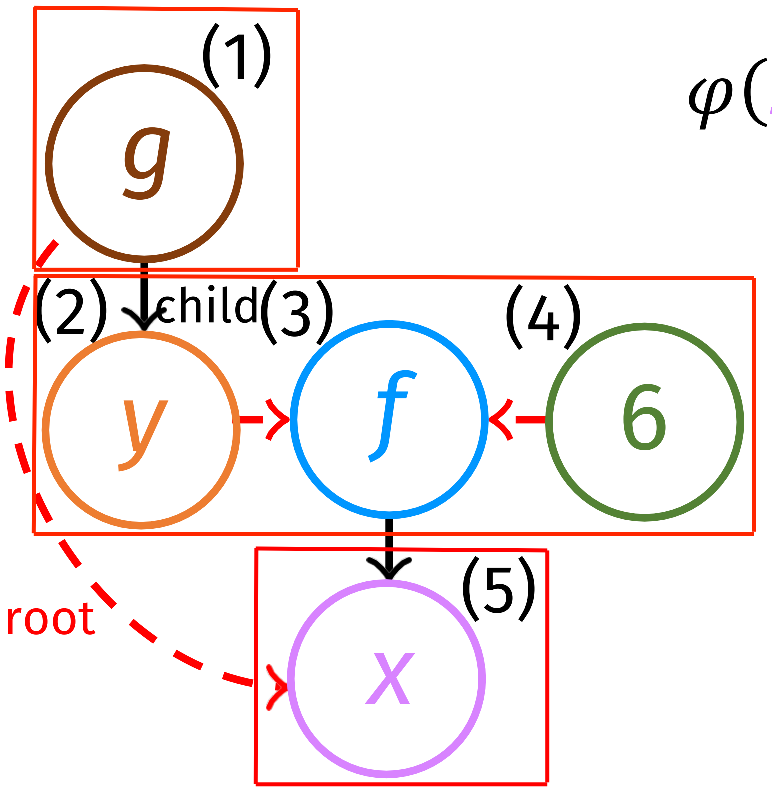
Our aim: fast quantifier reduction

Quickly try to remove variables (reduction of variables)

Consider all definitions

Egraphs!

What are egraphs?



$$\varphi(x, y) \triangleq \underset{(2)}{y} \approx \underset{(3)}{f(x)} \wedge \underset{(5)}{x} \approx \underset{(1)}{g(y)} \wedge f(x) \approx \underset{(4)}{6}$$

root(N(2)) = N(3)
root(N(4)) = N(5)
root(N(1)) = N(5)

class(N(1)) = {N(1), N(5)}
class(N(2)) = {N(3), N(2), N(4)}

Notation

$$G = egraph(\varphi)$$

Extracting terms from an egraph

Find one desired node per class → representative (rep)

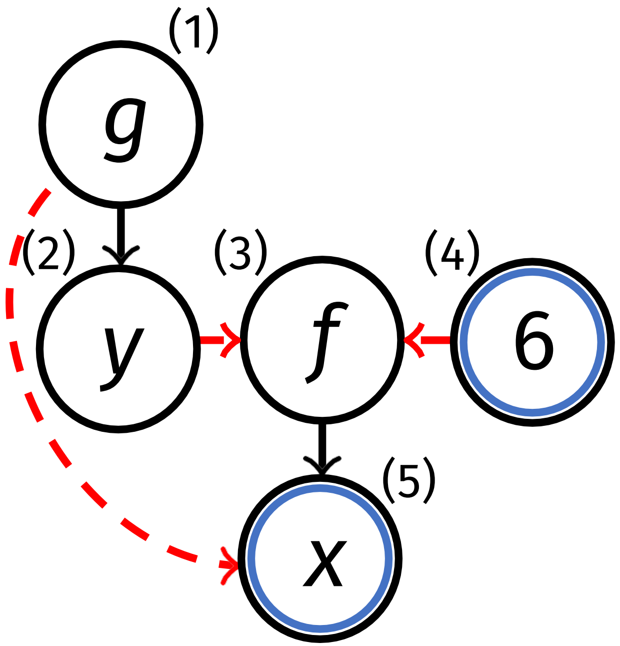
To extract a term of a node, use the terms of reps of its children

Notation

repr : N → N (representative function)
repr = {N(i), N(j)} (we describe rep functions by the set of representatives)

ntt(node,repr) (node-to-term, we omit repr if obvious)

Extracting terms from an egraph



Example repr = {N(4), N(5)}

repr(N(2)) = N(4)
repr(N(3)) = N(4)
repr(N(4)) = N(4)
repr(N(1)) = N(5)
repr(N(5)) = N(5)

ntt(N(5), repr) = x

ntt(N(3), repr) = $f(x)$

ntt(N(1), repr) = $g(6)$

Extracting formulas from an egraph

NEW!

Given a rep function repr, produce for each node:

$$\text{ntt}(\text{repr}(n)) \approx \text{ntt}(n)$$

Notation

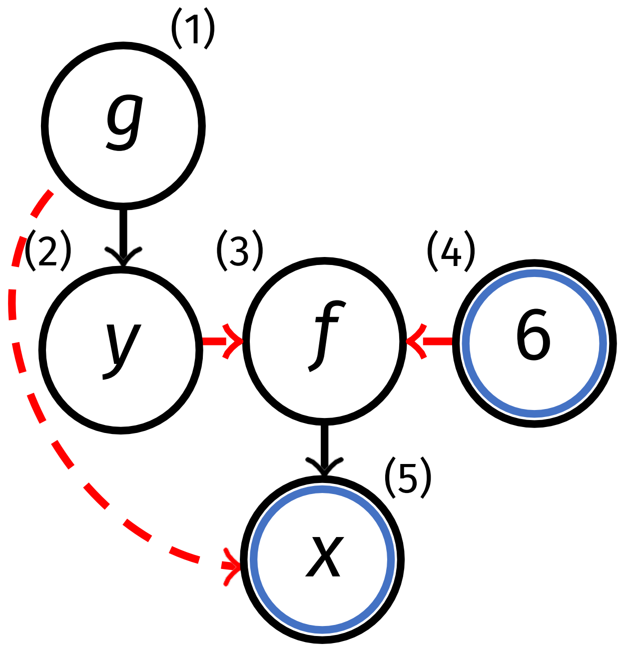
G.to_formula(repr)   (extract a formula from an egraph)

$(\,\bullet\,)^{\exists}$ existential closure

Guarantee : G = $egraph(\varphi)$, $\varphi^{\exists} \equiv (\text{G.to\_formula}(\text{repr}))^{\exists}$

Extracting formulas from an egraph

G = $egraph(y \approx f(x) \land x \approx g(y) \land f(x) \approx 6)$



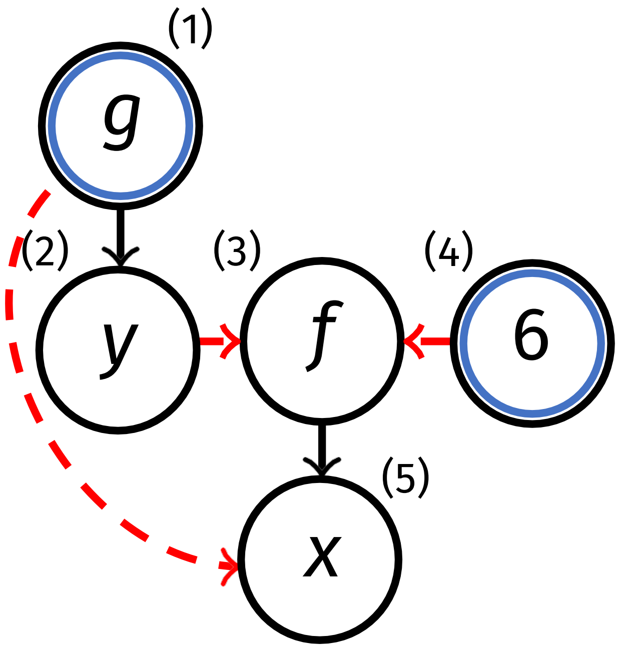
repr = {N(4), N(5)}

G.to_formula(repr) =

$$\underbrace{6 \approx f(x) \land 6 \approx y}_{\text{class(N(4))}} \land \underbrace{x \approx g(6)}_{\text{class(N(5))}}$$

Extracting for qelim

G = $egraph(y \approx f(x) \wedge x \approx g(y) \wedge f(x) \approx 6)$



repr = {N(4), N(1)}

G.to_formula(repr) =

$$\underbrace{\boxed{6 \approx f(g(6))} \wedge 6 \approx y}_{\text{class(N(4))}} \wedge \underbrace{g(f(6)) \approx x}_{\text{class(N(5))}}$$

bigger formula but more suitable for qelim!
just drop $(6 \approx y \wedge g(f(6)) \approx x)$

QEL – Quantifier reduction using egraphs

NEW!

Problem that we are trying to solve:

Given a quantifier free formula $A(\boldsymbol{v})$,

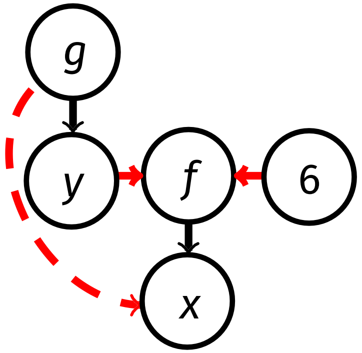
find $B(\boldsymbol{u})$ with $\boldsymbol{u} \subseteq \boldsymbol{v}$ and $B(\boldsymbol{u})^{\exists} \equiv A(\boldsymbol{v})^{\exists}$

if $\boldsymbol{u}$ is empty, we have qelim

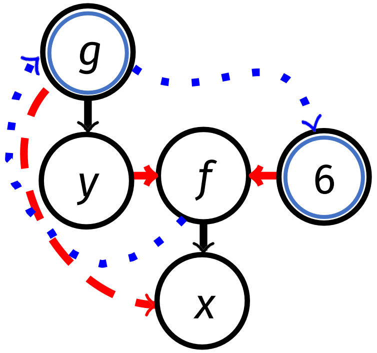
Using transitivity & congruence axioms

QEL – Quantifier reduction using egraphs

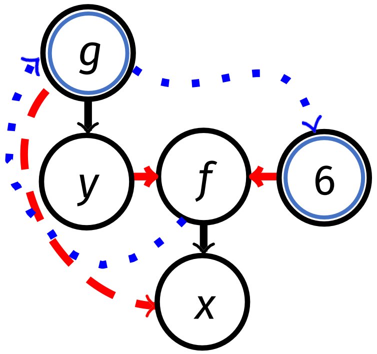
1. Build egraph



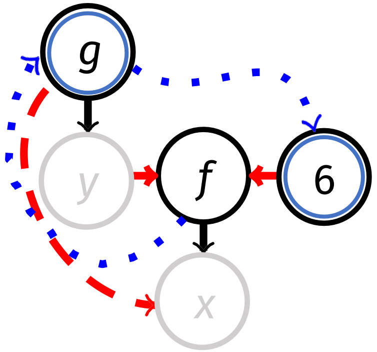
2. Find (ground) definitions



3. [Opt] Refine



4. Find a core

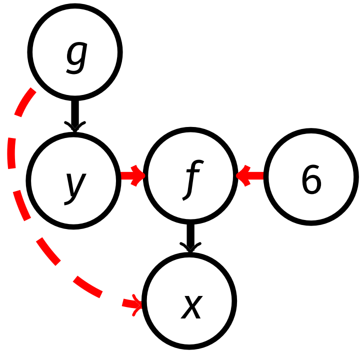


5. Output core

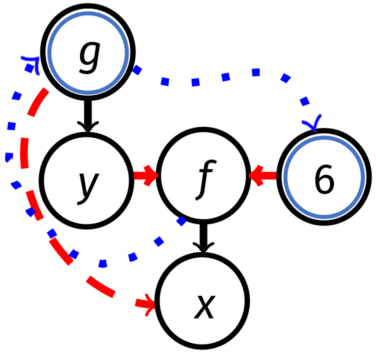
$$6 \approx f\big(g(6)\big)$$

QEL – Quantifier reduction using egraphs

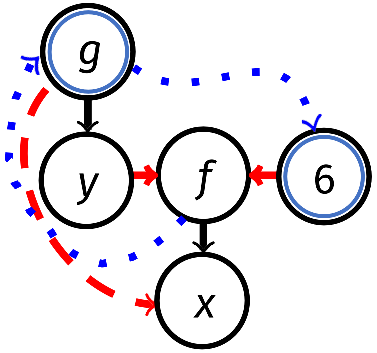
1. Build egraph



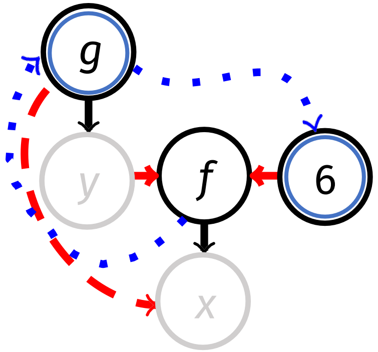
2. Find (ground) definitions



3. [Opt] Refine



4. Find a core



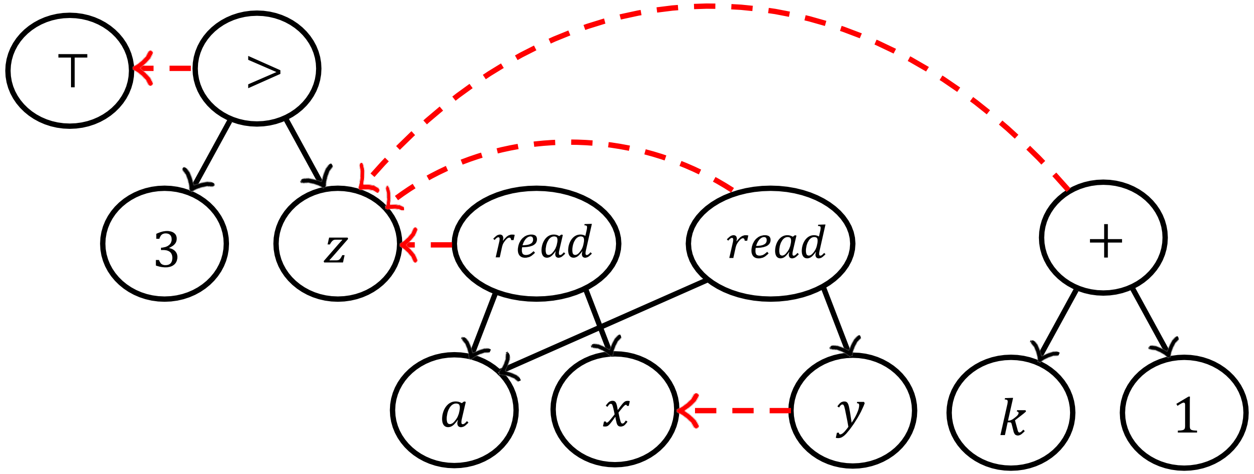
5. Output core

$$6 \approx f\big(g(6)\big)$$

Constructively ground

A class is *ground* if it contains a node that is constructively ground

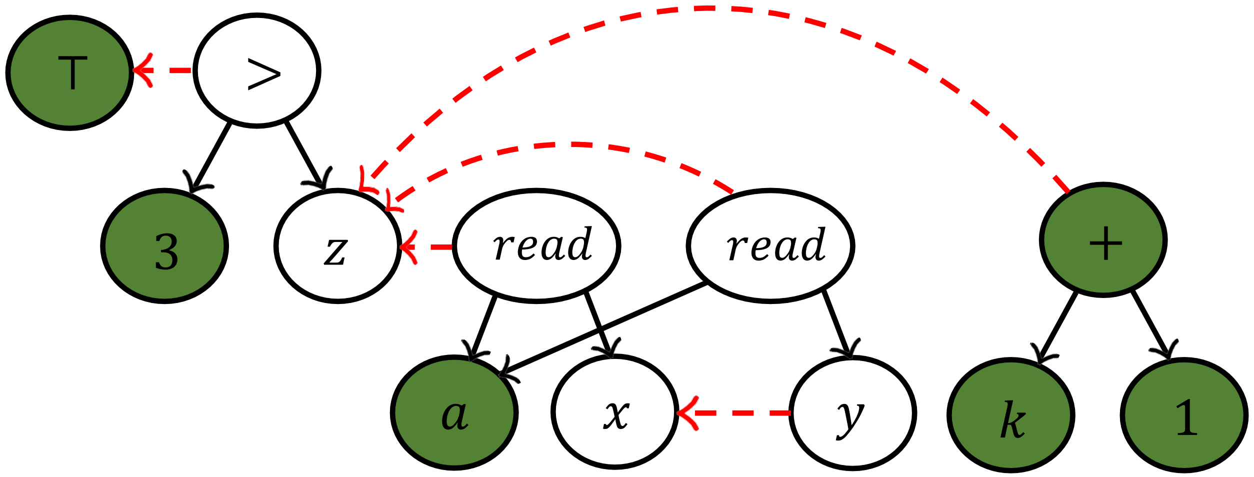
$$\gamma(x, y, z) \triangleq z \approx read(a, x) \wedge k + 1 \approx read(a, y) \wedge x \approx y \wedge 3 > z$$



Constructively ground

A class is *ground* if it contains a node that is constructively ground

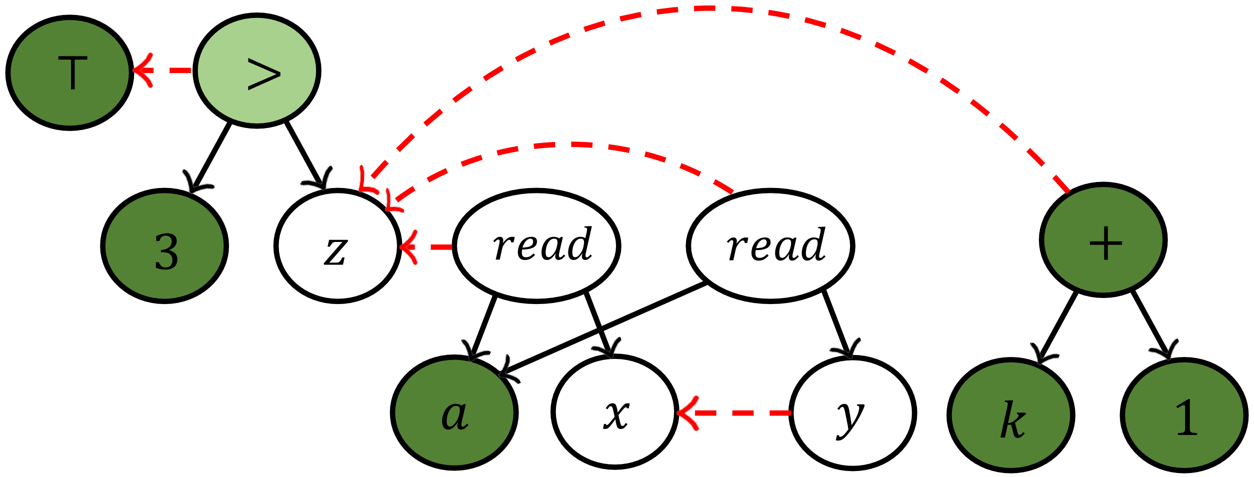
$$\gamma(x,y,z) \triangleq z \approx read(a,x) \land k+1 \approx read(a,y) \land x \approx y \land 3 > z$$



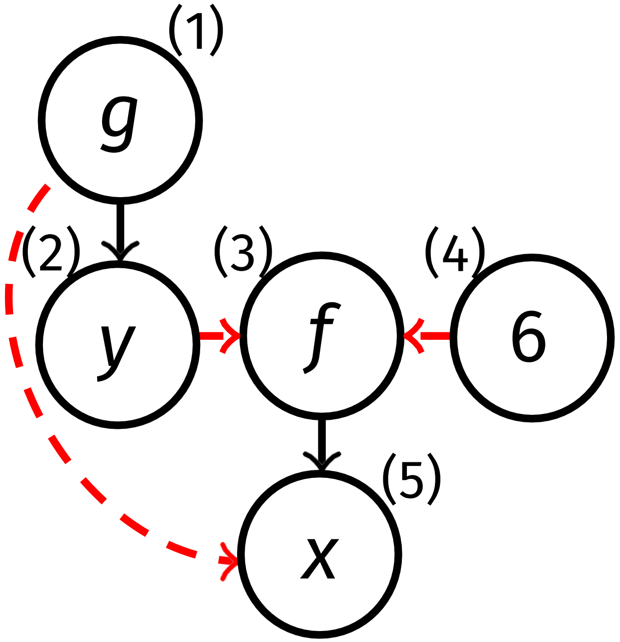
Constructively ground

A class is *ground* if it contains a node that is constructively ground

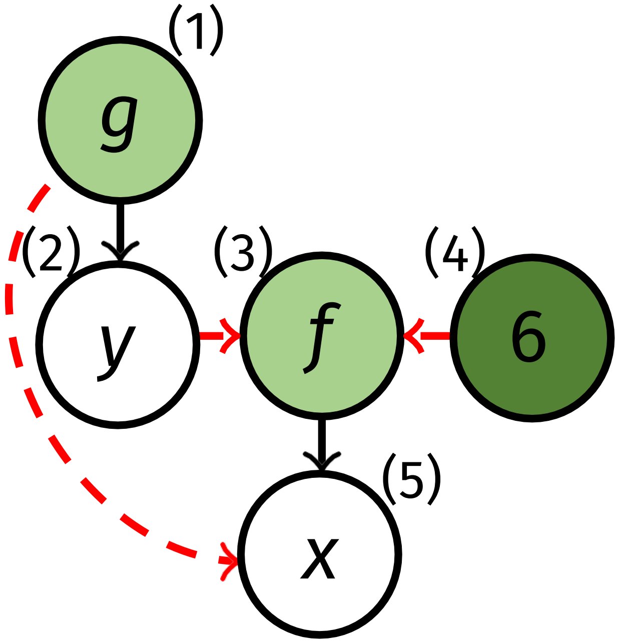
$$\gamma(x,y,z) \triangleq z \approx read(a,x) \wedge k+1 \approx read(a,y) \wedge x \approx y \wedge 3 > z$$



Find repr maximizing constr. ground

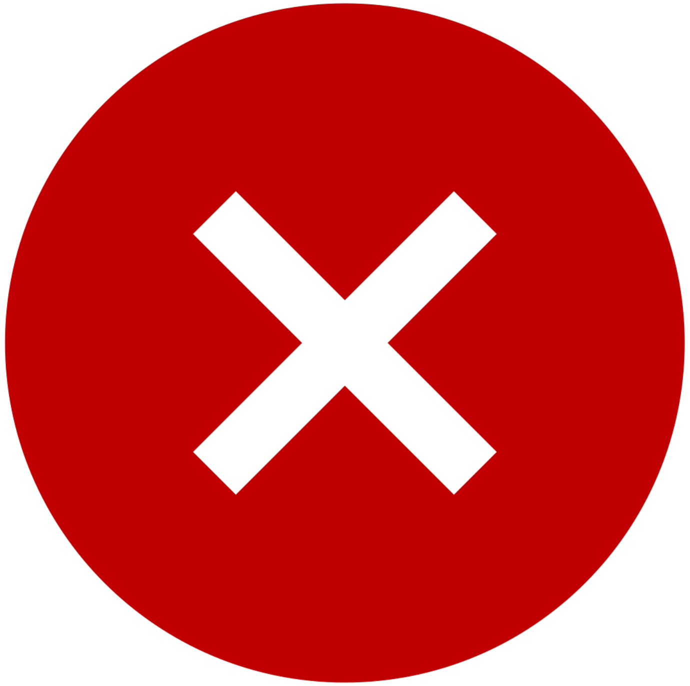


Find repr maximizing constr. ground



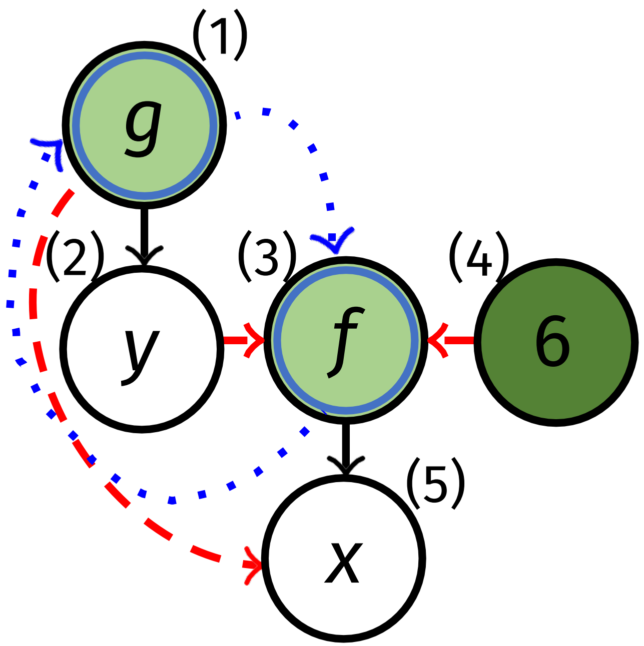
Let's choose constr. ground nodes as representatives!

Wait…
can we extract using any representative function?



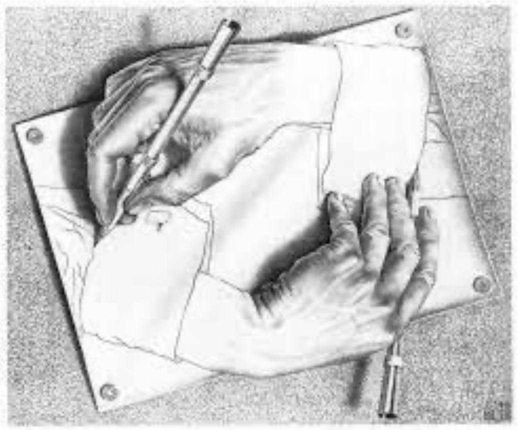
Not all representative functions guarantee that ntt-extraction terminates

Inadmissible representative function



Example repr = {N(1), N(3)}

ntt(N(1)) = ntt(g(ntt(N(3))) = ntt(g(f(ntt(N(1))) ...



Admissible representative functions

NEW!

A representative function `repr` is **admissible** iff:

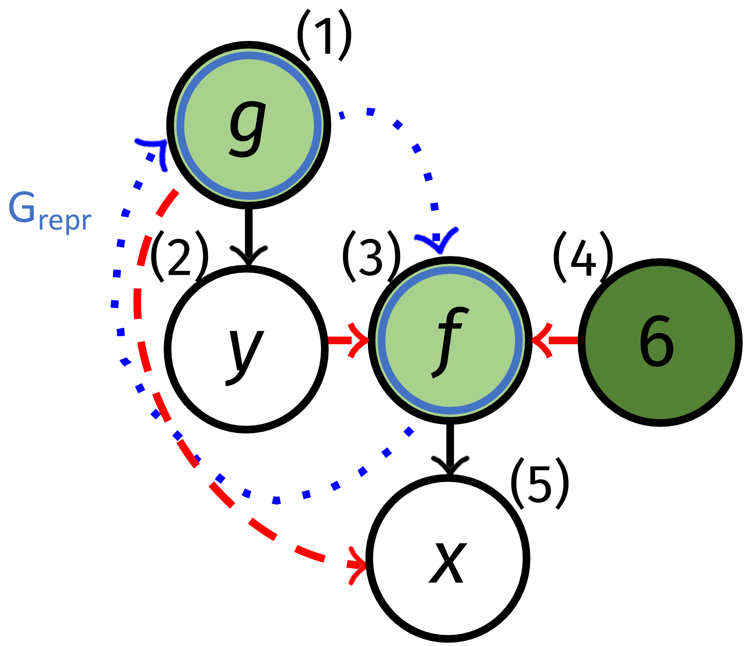
- **unique** rep per class
- `repr` defines the **same classes** as root
- a node is not a representative of any of its `repr`-descendants

Intuitively, the term of a node is not necessary to produce its own term

Formally, the graph $G_{repr} = \langle N, E_{repr} \rangle$ with

$$E_{repr} \triangleq \{ (n, repr(c)) \mid c \in children(n), n \in N \}$$ is acyclic

Admissible representative functions

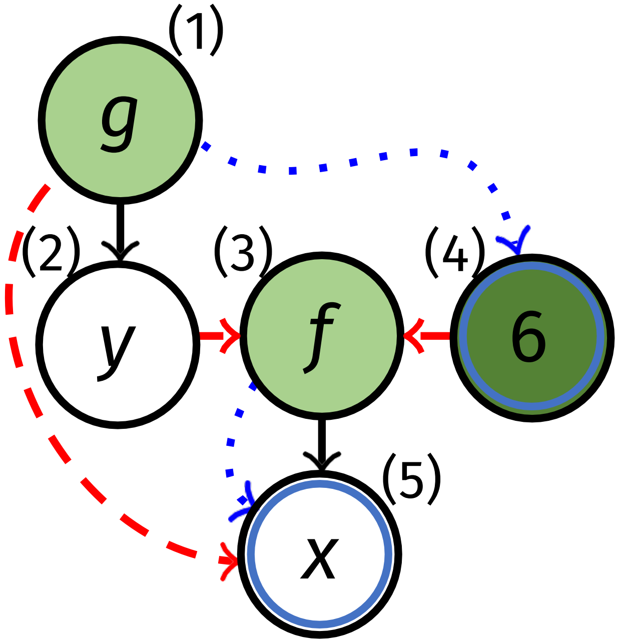


Example repr = {N(1), N(3)}

ntt(N(1)) = ntt(g(ntt(N(3))) = ntt(g(f(ntt(N(1))) …

Not admissible!

Admissible representative functions



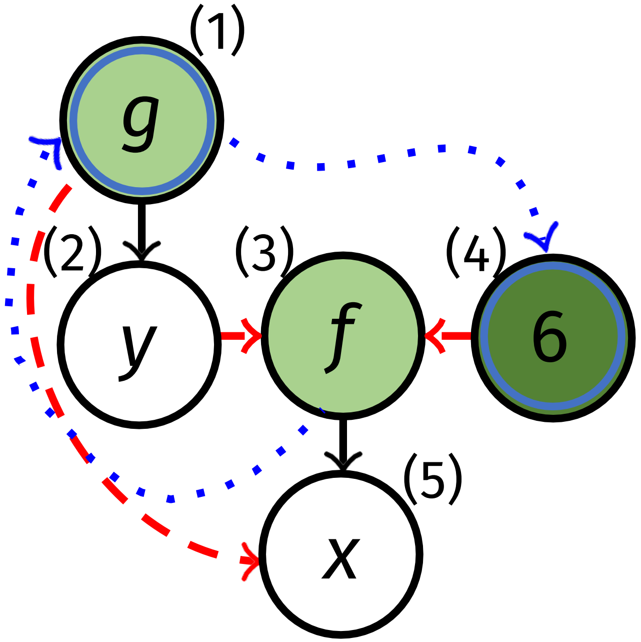
`repr` = {N(4), N(5)}

Admissible!

Admissibility is a *necessary* and *sufficient* condition for termination of `to_formula`

Choosing reps. based on (smaller) AST size guarantees admissibility…

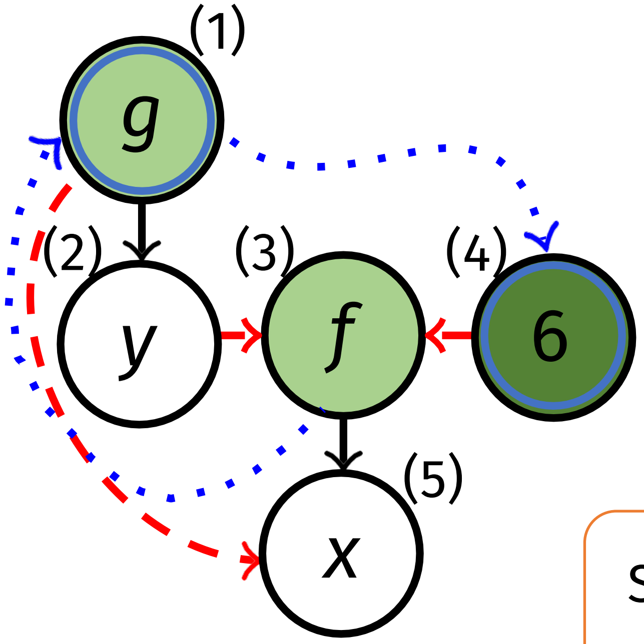
Find repr maximizing constr. ground



repr = {N(4), N(1)}

Admissible!

Find repr maximizing constr. ground



repr = {N(4), N(1)}

Why do constr. ground reps help?

Since terms of reps are used in ntt, variables with ground rep appear only once using to_formula:

G.to_formula(repr) = $6 \approx f\big(g(6)\big) \land 6 \approx y \land g(f(6)) \approx x$

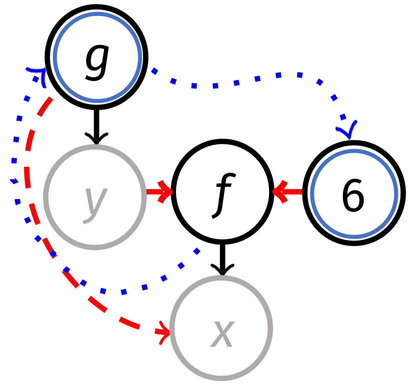
Easy elimination!

QEL guarantees

A variable is eliminated if:

- It has a ground definition
- Its node is not reachable in G_{repr} by any of the nodes in the core*

$$\varphi(x, y) \triangleq y \approx f(x) \land x \approx g(y) \land f(x) \approx 6$$



$$\varphi' \triangleq 6 \approx f(g(6))$$

qelim!

If all variables meet the conditions, we find a qelim

QEL is stronger than variable substitution

$\exists x \cdot x \approx g(f(x)) \land f(x) \approx 6$ QEL finds $6 \approx g(f(6))$

For $\exists x, y \cdot f(x) \approx f(y) \land x \approx y$, QEL produces $\top$, which is a qelim

Model-Based Projection

Under-approximation if variables were not eliminated

Example: $\exists x \cdot f(x) > 5$: a projection is $f(0) > 5$

Presented as rewrite rules

Guarantees:
- rewriting terminates
- result is an under-approximation
- output contains no variables

$$\textsc{ElimWrRd}\ \frac{\varphi[rd(wr(t,i,v),j)]}{(i = j \wedge \varphi[v]) \vee (i \neq j \wedge \varphi[rd(t,j)])}$$

$$\textsc{ElimWrEq}\ \frac{\varphi[wr(t_1,j,v) =_{\bar{i}} t_2]}{(j \in \bar{i} \wedge \varphi[t_1 =_{\bar{i}} t_2]) \vee \big(j \notin \bar{i} \wedge \varphi[t_1 =_{\bar{i},j} t_2 \wedge v = rd(t_2,j)]\big)}$$

$$\textsc{PartialEq}\ \frac{\varphi[t_1 = t_2]}{\varphi[t_1 =_{\emptyset} t_2]}\ t_i\text{'s have array sort}$$

$$\textsc{TrivEq}\ \frac{\varphi[t =_{\bar{i}} t]}{\varphi[\top]}$$

$$\textsc{Symm}\ \frac{\varphi[t_1 =_{\bar{i}} t_2]}{\varphi[t_2 =_{\bar{i}} t_1]}\ \begin{array}{l} t_2 \text{ is a write term} \\ \text{but } t_1 \text{ is not}\end{array}$$

no approx.

approximate (split) based on a model

$$\textsc{MbpLeft}\ \frac{\varphi[\psi_1 \vee \psi_2] \qquad M \models \varphi, \psi_1}{\varphi[\psi_1]}$$

$$\textsc{MbpRight}\ \frac{\varphi[\psi_1 \vee \psi_2] \qquad M \models \varphi, \psi_2}{\varphi[\psi_2]}$$

$$\textsc{MbpVac}\ \frac{\varphi[\psi_1 \vee \psi_2] \qquad M \models \varphi \qquad M \not\models \psi_1, \psi_2}{\varphi[\bot]}$$

Rules are defined for different theories separately

Implementing MBP using egraphs

Repeat until variables eliminated (out of the core*):

(1) Apply equivalence-preserving MBP rules until saturation

(2) Remove nodes from core based on rules

(3) If there are variables, apply model-splitting MBP rules

Apply only to not constr. ground nodes in the core

Update constr. groundness

Very easy to combine theories, just as for SMT solving!

We implemented for ADTs and Arrays

Full elimination is guaranteed (under-approx.)

Implementation & evaluation – QSAT

Solving formulas alternating exists and forall quantifiers

Category	Count	Z3EG		Z3		YicesQS	
		SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
LIA	416	**150**	**266**	**150**	**266**	107	102
LRA	2 419	795	1 589	793	1 595	**808**	**1610**

Playing with Quantified Satisfaction

Nikolaj Bjørner[1] and Mikoláš Janota[2]

[1] Microsoft Research, Redmond, USA
[2] Microsoft Research, Cambridge, UK

Abstract

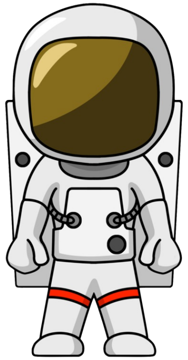
We develop an algorithm for satisfiability of quantified formulas. The algorithm is based on recent progress in solving Quantified Boolean Formulas, but it generalizes beyond propositional logic to theories, such as linear arithmetic over integers (Presburger arithmetic), linear arithmetic over reals, algebraic data-types and arrays. Compared with previous algorithms for satisfiability of quantified arithmetical formulas our new implementation outperforms previous implementations in Z3 by a significant margin.

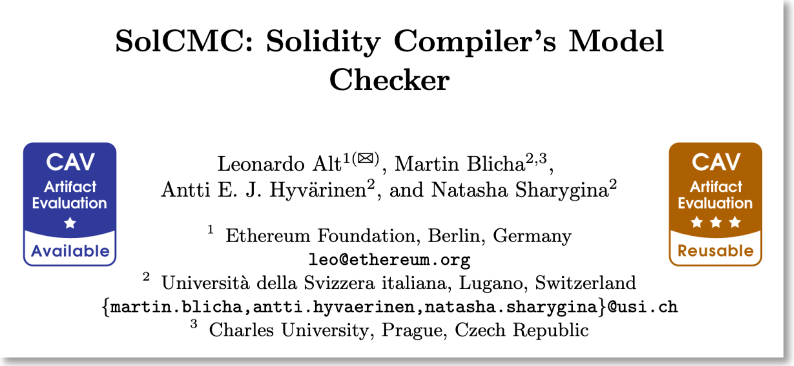
Category	Count	Z3EG		Z3	
		SAT	UNSAT	SAT	UNSAT
LIA-ADT	416	**150**	**266**	**150**	56
LRA-ADT	2 419	**757**	**1 415**	793	964

Implementation & evaluation – Spacer

CHC solving over ADTs, LIA and Arrays

Category	Count	Z3EG		Z3		ELDARICA	
		SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
Solidity	3 468	2 324	1 133	2 314	1 114	**2 329**	**1 134**
→ abi	127	19	108	19	88	19	108
LIA-lin-Arrays	488	**214**	72	212	**75**	147	68

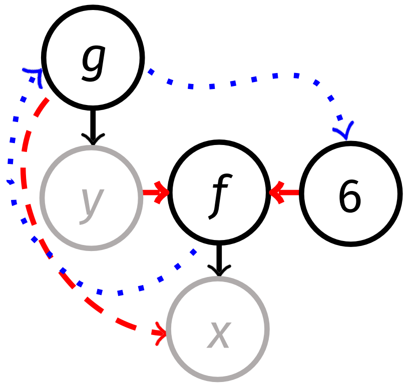



SolCMC: Solidity Compiler's Model Checker

Leonardo Alt[1(✉)], Martin Blicha[2,3], Antti E. J. Hyvärinen[2], and Natasha Sharygina[2]

[1] Ethereum Foundation, Berlin, Germany
leo@ethereum.org
[2] Università della Svizzera italiana, Lugano, Switzerland
{martin.blicha,antti.hyvaerinen,natasha.sharygina}@usi.ch
[3] Charles University, Prague, Czech Republic

Conclusion

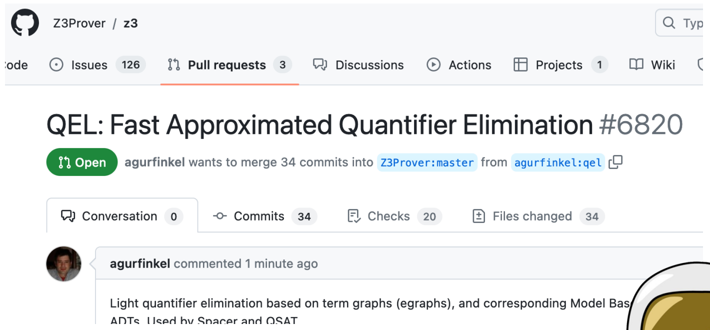


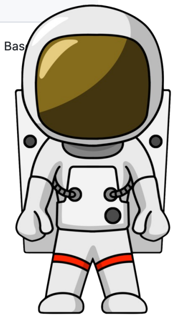
Characterized all possible extractions from egraphs via **admissible** representative functions

Presented QEL, an algorithm for quantifier reduction that is complete **relative** to ground definitions entailed by formulas

Use theory rewrites to under-approximate formulas (Model-Based Projection) when QEL was not complete

Implemented MBP for ADTs and Arrays





Implemented and evaluated within Z3: We used it to improve QSAT and the Spacer CHC solver

Fast Approximations of Quantifier Elimination

Isabel Garcia-Contreras
Hari Govind V K
Sharon Shoham
Arie Gurfinkel

Yes, I am the representative!

@CAV 2023

TEL AVIV UNIVERSITY אוניברסיטת תל אביב

UNIVERSITY OF WATERLOO

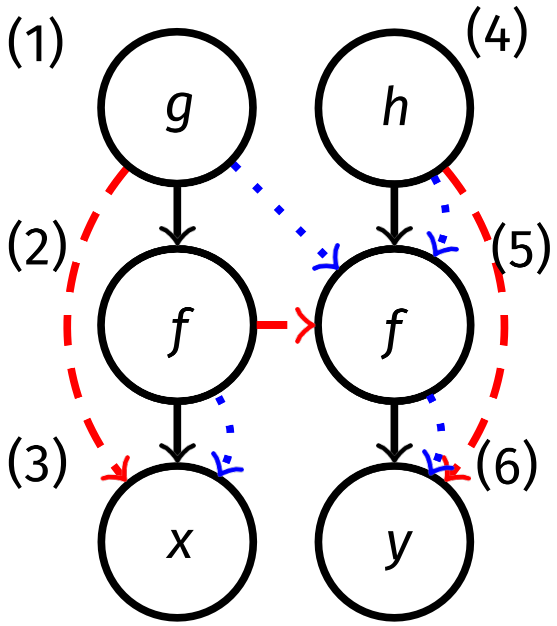
Thanks!

Refining repr

$$\psi(x,y) \triangleq x \approx g(f(x)) \wedge y \approx h(f(y)) \wedge f(x) \approx f(y)$$

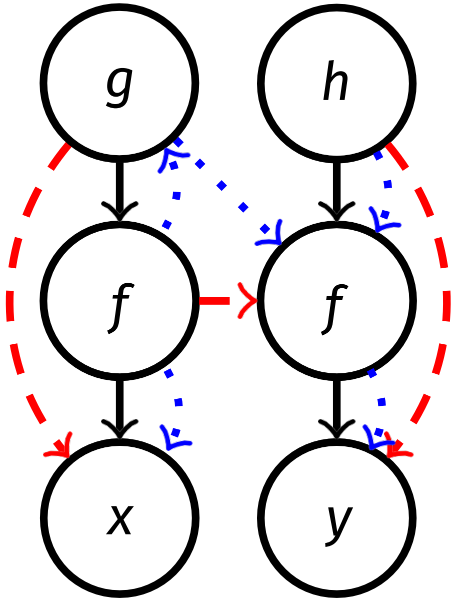
Nothing constr. ground

`repr` = {N(3), N(5), N(6)}



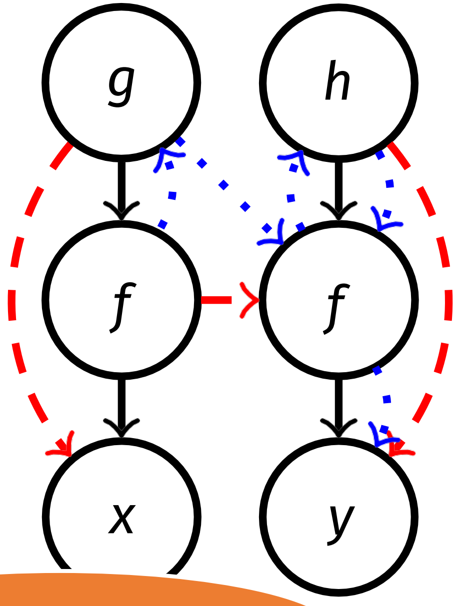
refine

`repr` = {N(1), N(5), N(6)}



try refine

`repr` = {N(1), N(5), N(4)}



Not admissible!

3 A Quantified Satisfiability Game

Algorithm 1: QSAT

$j \leftarrow 1$;
$M \leftarrow null$;
while *True* **do**
    if $F_j \wedge strategy(M, j)$ *is unsat* **then**
        if j = *1* **then**
            return G is false
        if j = *2* **then**
            return G is true
        $C \leftarrow Core(F_j, strategy(M, j))$;
        $J \leftarrow Mbp(M, tail(j), C)$;
        $j \leftarrow$ index of max variable in $J \cup \{1, 2\}$ of same parity as j;
        $F_j \leftarrow F_j \wedge \neg J$;
        $M \leftarrow null$;
    else
        $M \leftarrow$ the current model;
        $j \leftarrow j + 1$;

Playing with Quantified Satisfaction

Nikolaj Bjørner[1] and Mikoláš Janota[2]

[1] Microsoft Research, Redmond, USA
[2] Microsoft Research, Cambridge, UK

Abstract

We develop an algorithm for satisfiability of quantified formulas. The algorithm is based on recent progress in solving Quantified Boolean Formulas, but it generalizes beyond propositional logic to theories, such as linear arithmetic over integers (Presburger arithmetic), linear arithmetic over reals, algebraic data-types and arrays. Compared with previous algorithms for satisfiability of quantified arithmetical formulas our new implementation outperforms previous implementations in Z3 by a significant margin.

Bonus: Formulas with Minimal Variables Appearing?

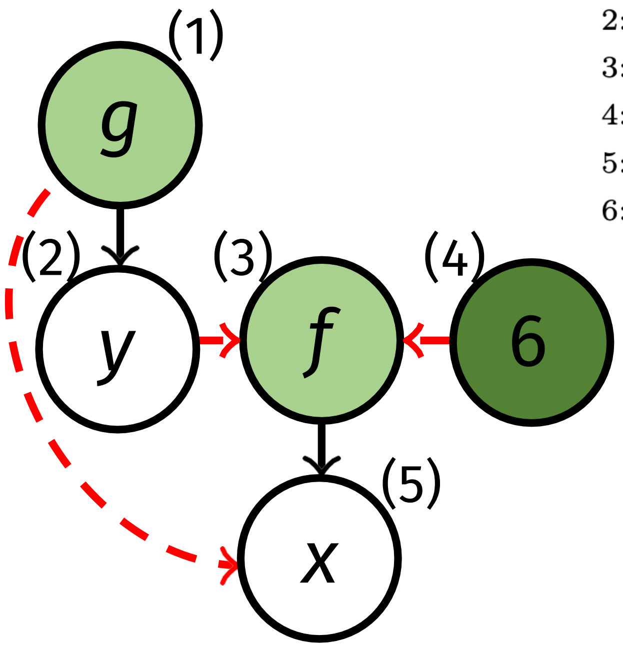
$$\varphi(x, y, z) \triangleq x \approx f(z) \land y \approx g(z) \land z \approx h(x, y)$$

Possible outputs, depending on refinement order of repr:

$$\varphi_1(z) \triangleq z \approx h(f(z), g(z))$$
$$\varphi_2(x, y) \triangleq x \approx f(h(x, y)) \land y \approx g(h(x, y))$$

Open question: hard due to sharing?

Find repr maximizing constr. ground



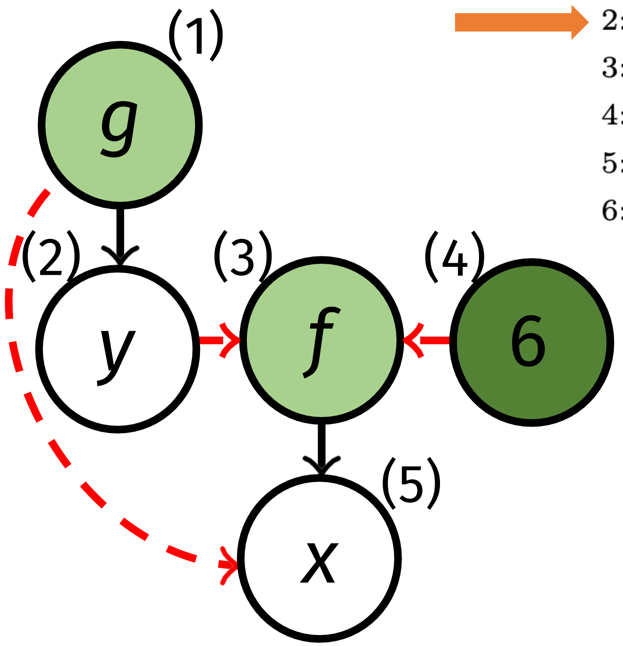
$egraph$ :: `find_defs`($\boldsymbol{v}$)

for $n \in N$ **do** `repr`(n) := ★
$todo := \{leaf(n) \mid n \in N \wedge ground(n)\}$
`repr` := `process`(`repr`, $todo$)
$todo := \{leaf(n) \mid n \in N\}$
`repr` := `process`(`repr`, $todo$)
ret `repr`

$egraph$ :: `process`(`repr`, $todo$)

while $todo \neq \emptyset$ **do**
  $n := todo.pop()$
  if `repr`(n) ≠ ★ **then continue**
  for $n' \in class(n)$ **do** `repr`(n') := n
  for $n' \in class(n)$ **do**
    for $p \in$ `parents`(n') **do**
      if $\forall c \in$ `children`(p) · `repr`(c) ≠ ★ **then**
        $todo.push(p)$
ret `repr`

Find repr maximizing constr. ground



egraph :: find_defs(v)

for $n \in N$ **do** repr(n) := ★
todo := {*leaf*(n) | $n \in N \wedge$ *ground*(n)}
repr := process(repr, *todo*)
todo := {*leaf*(n) | $n \in N$}
repr := process(repr, *todo*)
ret repr

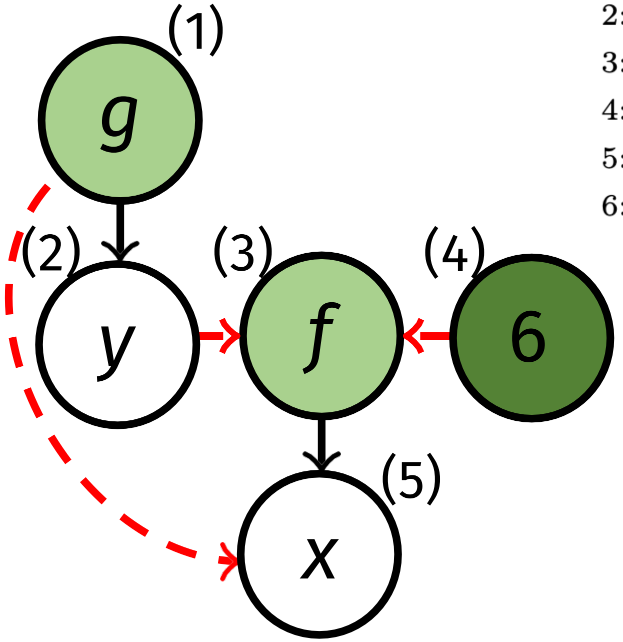
egraph :: process(repr, *todo*)

while *todo* $\neq \emptyset$ **do**
  n := *todo.pop*()
  if repr(n) $\neq$ ★ **then continue**
  for $n' \in$ *class*(n) **do** repr(n') := n
  for $n' \in$ *class*(n) **do**
    for $p \in$ parents(n') **do**
      if $\forall c \in$ children(p) $\cdot$ repr(c) $\neq$ ★ **then**
        todo.push(p)
ret repr

todo = [N(4)]

repr = {}

Find repr maximizing constr. ground



egraph :: find_defs(v)

```
for n ∈ N do repr(n) := ★
todo := {leaf(n) | n ∈ N ∧ ground(n)}
repr := process(repr, todo)
todo := {leaf(n) | n ∈ N}
repr := process(repr, todo)
ret repr
```

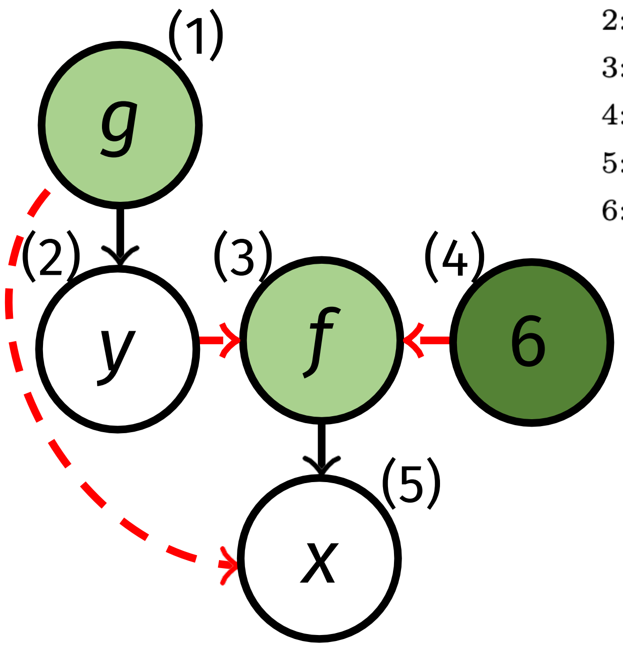
egraph :: process(repr, *todo*)

```
while todo ≠ ∅ do
  n := todo.pop()
  if repr(n) ≠ ★ then continue
  for n′ ∈ class(n) do repr(n′) := n
  for n′ ∈ class(n) do
    for p ∈ parents(n′) do
      if ∀c ∈ children(p) · repr(c) ≠ ★ then
        todo.push(p)
ret repr
```

todo = []

repr = {N(4)}

Find repr maximizing constr. ground



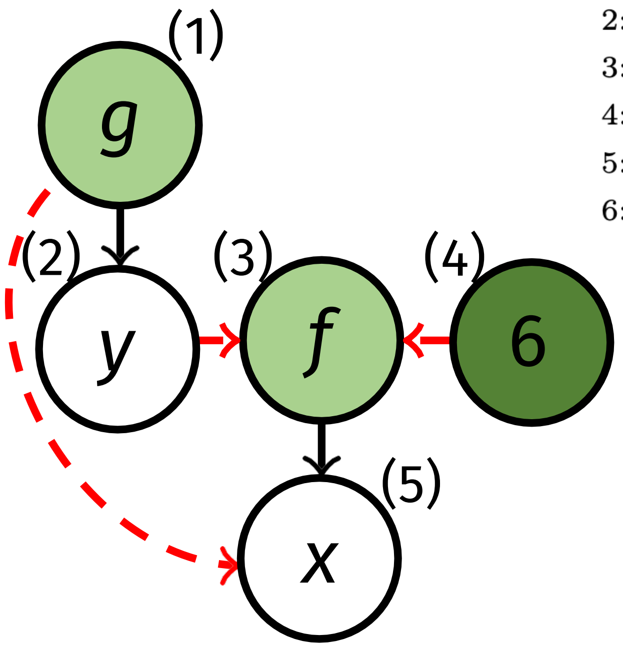

```
egraph :: find_defs(v)
for n ∈ N do repr(n) := ★
todo := {leaf(n) | n ∈ N ∧ ground(n)}
repr := process(repr, todo)
todo := {leaf(n) | n ∈ N}
repr := process(repr, todo)
ret repr
```

```
egraph :: process(repr, todo)
while todo ≠ ∅ do
  n := todo.pop()
  if repr(n) ≠ ★ then continue
  for n′ ∈ class(n) do repr(n′) := n
  for n′ ∈ class(n) do
    for p ∈ parents(n′) do
      if ∀c ∈ children(p) · repr(c) ≠ ★ then
        todo.push(p)
ret repr
```

todo = [N(1)]

repr = {N(4)}

Find repr maximizing constr. ground



$egraph :: \texttt{find\_defs}(\boldsymbol{v})$

- **for** $n \in N$ **do** $\texttt{repr}(n) := \bigstar$
- $todo := \{leaf(n) \mid n \in N \wedge ground(n)\}$
- $\texttt{repr} := \texttt{process}(\texttt{repr}, todo)$
- $todo := \{leaf(n) \mid n \in N\}$
- $\texttt{repr} := \texttt{process}(\texttt{repr}, todo)$
- **ret** $\texttt{repr}$

$egraph :: \texttt{process}(\texttt{repr}, todo)$

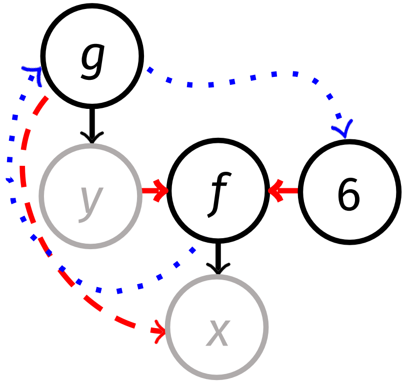
- **while** $todo \neq \emptyset$ **do**
  - $n := todo.pop()$
  - **if** $\texttt{repr}(n) \neq \bigstar$ **then continue**
  - **for** $n' \in class(n)$ **do** $\texttt{repr}(n') := n$
  - **for** $n' \in class(n)$ **do**
    - **for** $p \in \texttt{parents}(n')$ **do**
      - **if** $\forall c \in \texttt{children}(p) \cdot \texttt{repr}(c) \neq \bigstar$ **then**
        - $todo.push(p)$
- **ret** $\texttt{repr}$

todo = []

repr = {N(4), N(1)}

QEL examples

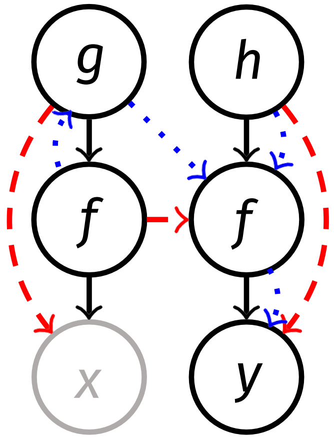
$\varphi(x,y) \triangleq y \approx f(x) \wedge x \approx g(y) \wedge f(x) \approx 6$



$\varphi' \triangleq 6 \approx f(g(6))$

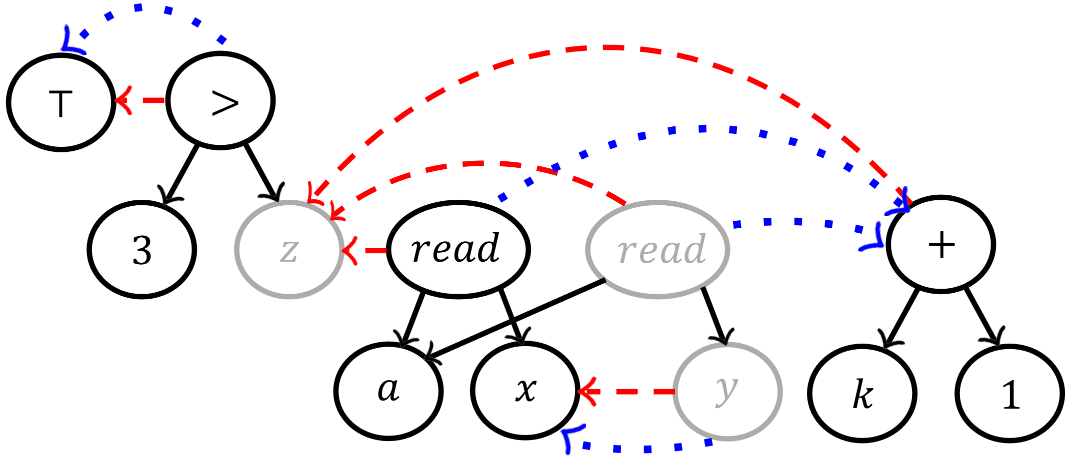
qelim!

$\psi(x,y) \triangleq x \approx g(f(x)) \wedge y \approx h(f(y)) \wedge f(x) \approx f(y)$



$\psi'(y) \triangleq y \approx h(f(y)) \wedge f(y) \approx f(g(f(y)))$

$\gamma(x,y,z) \triangleq z \approx read(a,x) \wedge k+1 \approx read(a,y) \wedge x \approx y \wedge 3 > z$



$\gamma'(x) \triangleq k+1 \approx read(a,x) \wedge 3 > k+1$