Engineering Satisfiability Modulo Theories Solvers for Intractable Problems

Nikolaj Bjørner
Microsoft Research

Tractability Workshop – MSR Cambridge July 5,6 2010

FSE & RiSE

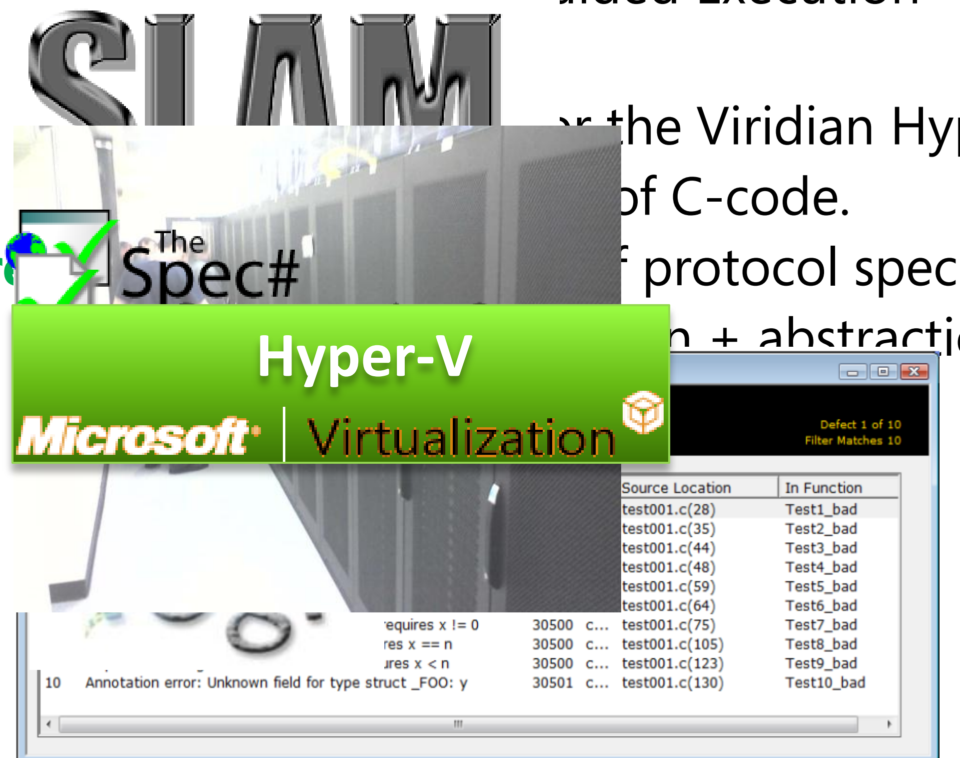
This talk

Z3 – An Efficient SMT solver:
Overview and Applications.

A “hands on” example of Engineering SMT solvers:
Efficient Theory Resolution using DPLL(T).

Some Microsoft Engines using Z3

- **SDV:** The Static Driver Verifier
- **PREfix:** The Static Analysis Engine for C/C++.
- **Pex:** Program EXploration for .NET.
- **SAGE:** Scalable Automated Guided Execution
- **Spec#:**
- **VCC:** ...r the Viridian Hyper-Visor
- **HAVOC:** ...of C-code.
- **SpecExplor...** ...f protocol specs.
- **Yogi:** ...n + abstraction.
- **FORMULA:**
- **F7:**
- **M3:**
- **VS3:**
- **VERVE:**
- **FINE:**



SAGE by the numbers

Slide shamelessly stolen and adapted from [Patrice Godefroid, ISSTA 2010]

100+ CPU-years - largest dedicated fuzz lab in the world

100s apps - fuzzed using SAGE

100s previously unknown bugs found

1,000,000,000+ computers updated with bug fixes

Millions of **$** saved for Users and Microsoft

10s of related tools (incl. Pex), 100s DART citations

100,000,000+ constraints - largest usage for any SMT solver

PREfix [Moy, B., Sielaff]

3(INT_MAX+1)/4 +
(INT_MAX+1)/4
= INT_MIN

-INT_MIN=
INT_MIN

```
int binary_se
while (low <= hig
  {
     // Find middle value
     int mid = (low + high) / 2;
     int val = arr[mid];
     if (val == key) return mid;
     if (val < key) low = mid+1;
     else high = mid-1;
  }
  return -1;
```

Package: java.util.Arrays
Function: binary_search

```
id itoa(int n, char*
  if (n < 0) {
     *s++ = '-';
     n = -n;
  }
  // Add digits to s
  ....
```

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)

Example: an overflowed allocation size

Overflow check

```
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
```

Increment and exit from loop

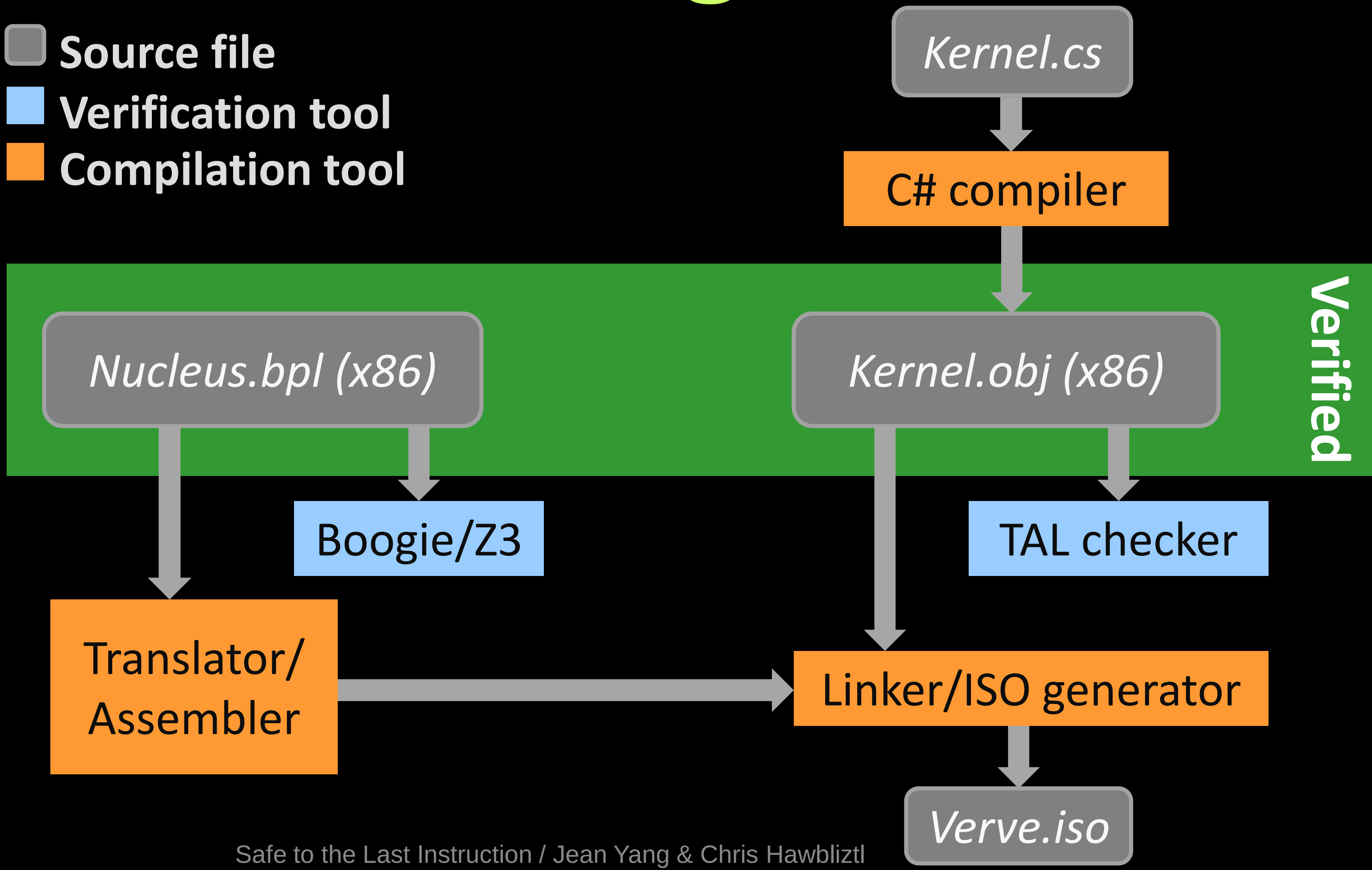
Possible overflow

Bug is simple and local within a large program

…
Overflow((nb+1)*sizeof(MYBUFFER))
CurrentBuffer == NULL
nb <= MAX_ULONG/sizeof(MYBUFFER)

Building Verve

9 person-months



What is Satisfiability Modulo Theories?

$$\boxed{x+2=y} \Rightarrow \boxed{f}(\boxed{read}(\boxed{write}(a,x,\boxed{3}),\boxed{y-2})) = \boxed{f}(\boxed{y-x+1})$$

Array Theory

Arithmetic

Uninterpreted Functions

$$read(write(a,i,v),i) = v$$

$$i \neq j \Rightarrow read(write(a,i,v),j) = read(a,j)$$

What is Z3?

Simplify

SMT-LIB

Native

Theory Solvers

Bit-Vectors

Arrays

Lin-arithmetic

Groebner basis

Recursive Datatypes

Comb. Array Logic

Free (uninterpreted) functions

OCaml

.NET

C

F# quote

SAT core

Model Generation: Finite Models

Quantifiers: Super-position

Proof objects

Quantifiers: E-matching

Parallel Z3

Cores: Assumption tracking

By Leonardo de Moura & Nikolaj Bjørner http://research.microsoft.com/projects/z3

Tractability and Applications

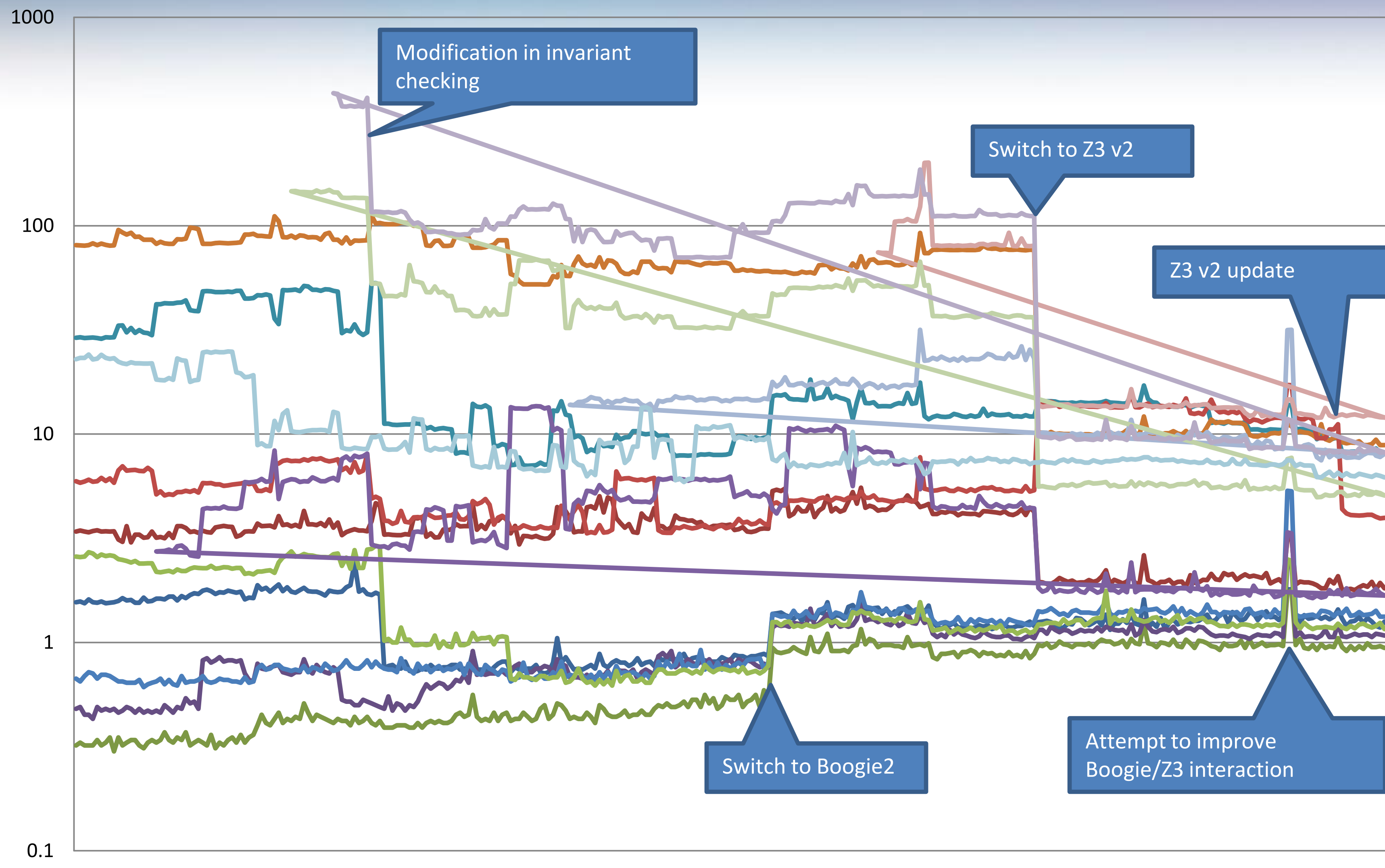
Constraints from Software Applications are

in spite of

Constraint language	highly intractable
Algorithms	high worst case complexity

Tractable

VCC Performance Trends Nov 08 – Mar 09



The Importance of Speed

Subject: FW: Der neue Z3 ist höllisch schnell (und ich meine kein Auto)

Fyi.

Wi

Translator

Home | Tools | Help

Free online translation service for a truly *worldwide* web

Fro
Se
To:
Sul

Hal

Languages German → English Translate Clear All Add to Favorites

Enter text or webpage URL

Report offensive translations

Ich habe einmal den neuen VCC auf mein Beispiel losgelassen, das ansonsten erst nach 50000 Sekunden irgendein Ergebnis produziert hat. Nun erhalte ich die ersten Fehler schon nach 200-300 Sekunden. Von daher bin ich sehr glücklich und zufrieden! Das ist gewaltiger Fortschritt.

I have released the new VCC once on my example has produced any result otherwise after 50000 seconds. Now, I receive the first error already after 200-300 seconds. That is why I am very happy and satisfied! This is huge progress.

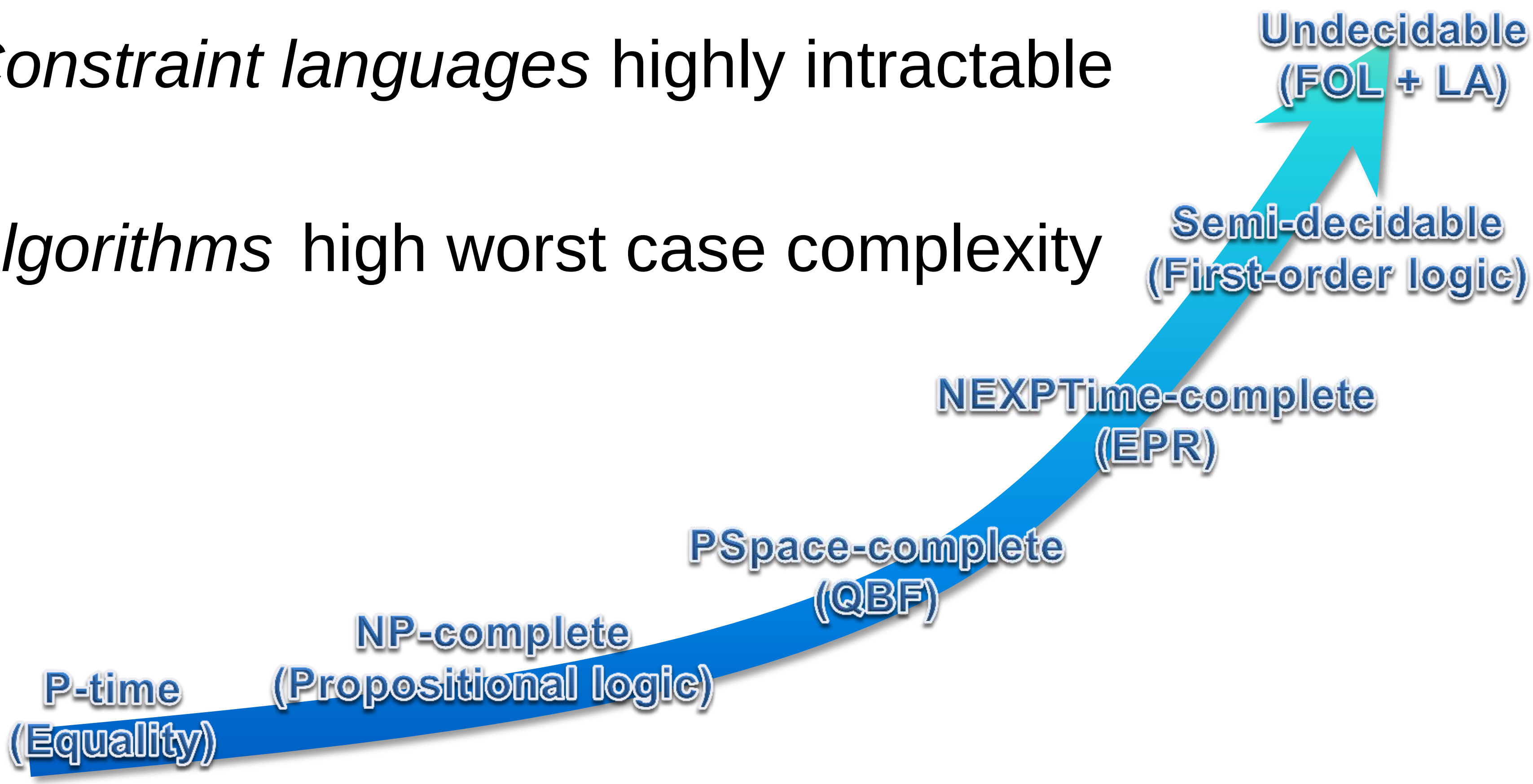
Ich habe einmal den neuen VCC auf mein Beispiel losgelassen, das ansonsten erst nach 50000 Sekunden irgendein Ergebnis produziert hat. Nun erhalte ich die ersten Fehler schon nach 200-300 Sekunden. Von daher bin ich sehr glücklich und zufrieden! Das ist gewaltiger Fortschritt.

Viel Spaß und liebe Grüße an Lieven,
Markus

Tractability and Applications

Constraint languages highly intractable

Algorithms high worst case complexity



Tractability and Applications

Constraints from Software Applications are Tractable

$$\left.\begin{array}{l} a \leq b \wedge \\ b < c \wedge \\ c \leq a \wedge \end{array}\right\} \textbf{Unsat}$$

$$\begin{array}{l} x \leq y \wedge \\ y < z \wedge \\ z < u \wedge \\ x \leq w \wedge \\ x \leq v \wedge \\ x \leq 1 \wedge \\ x \leq 2 \wedge \\ x \leq 3 \end{array}$$

Proofs are small

$$\left.\begin{array}{l} a \leq b \wedge \\ b \leq c \wedge \\ c \leq a \wedge \end{array}\right\} \boldsymbol{a = b = c}$$

$$\left.\begin{array}{l} x = w \wedge \\ x = v \wedge \\ x = 1 \wedge \end{array}\right\} \boldsymbol{x, v, w = 1}$$

$$\left.\begin{array}{l} x \leq 2 \wedge \\ x \leq 3 \wedge \end{array}\right\} \boldsymbol{x = 1 \leq 2,3}$$

$$\left.\begin{array}{l} x \leq y \wedge \\ y < z \wedge \\ z < u \wedge \end{array}\right\} \boldsymbol{y,z,u \text{ “free”}}$$

Models are determined or free

Tractability and Applications

What is then important for engineering solvers?

Solve tractable parts	- efficient theory solvers
Strong Simplification	- reduce the clutter
Efficient Indexing	- minimize & reuse work
Avoid getting stuck	- restarts, parallel search

Tractability and Applications

What is then important for engineering solvers?

Solve tractable parts	- efficient theory solvers
	[Efficient, Generalized Array Decision Procedures de Moura & B]
Strong Simplification	- reduce the clutter
	[Z3 An Efficient SMT Solver de Moura & B]
Efficient Indexing	- minimize & reuse work
	[Efficient E-matching de Moura & B]
Avoid getting stuck	- restarts, parallel search
	[Parallel Portfolio, Wintersteiger, Hamadi & de Moura]

Tractability and Applications

Constraints from Software Applications are Tractable

Problem solved, end of talk

Tractability and Applications

Constraints from Software Applications are Tractable

sometimes quite intractable for existing techniques

Symptom of a problem

```
public void Diamond(int a) {
  if (p1(a))
       a++;
   else
       a--;
  }
  ...
  if (p100(a))
       a++;
   else
       a--;
  }
```

$$\left[\begin{array}{l} \left(\begin{array}{l} \quad (p_1(a_0) \wedge a_1 \simeq a_0 + 1) \\ \vee\ (\neg p_1(a_0) \wedge a_1 \simeq a_0 - 1) \end{array}\right) \\ \wedge \left(\begin{array}{l} \quad (p_2(a_1) \wedge a_2 \simeq a_1 + 1) \\ \vee\ (\neg p_2(a_1) \wedge a_2 \simeq a_1 - 1) \end{array}\right) \\ \wedge \ldots \\ \wedge \left(\begin{array}{l} \quad (p_{100}(a_{99}) \wedge a_{100} \simeq a_{99} + 1) \\ \vee\ (\neg p_{101}(a_{99}) \wedge a_{100} \simeq a_{99} - 1) \end{array}\right) \end{array}\right]$$

$$\rightarrow$$

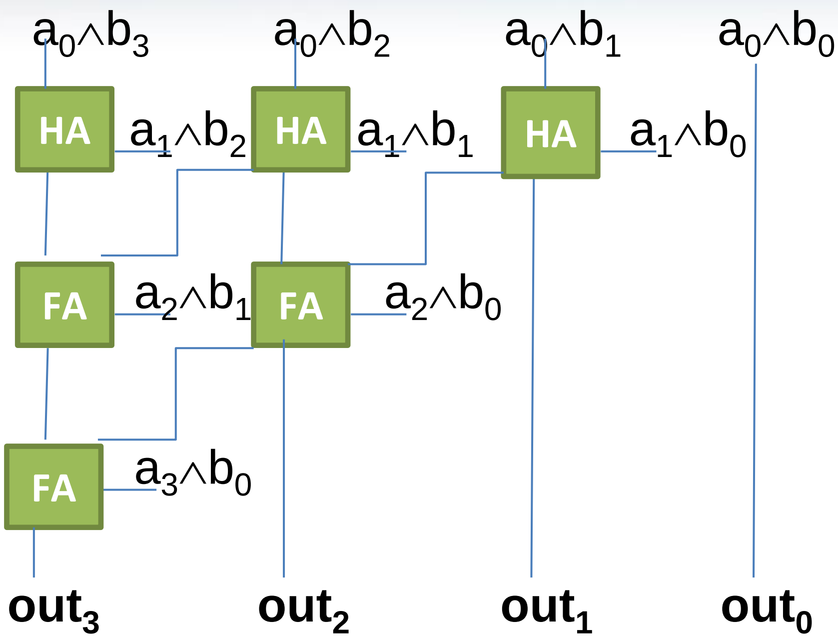
$$a_0 - 100 \leq a_{100} \leq a_0 + 100$$

$$assert(\mathrm{old}(a) - 100 \leq a \leq \mathrm{old}(a) + 100);$$

Poses a challenge to Z3

Another challenge

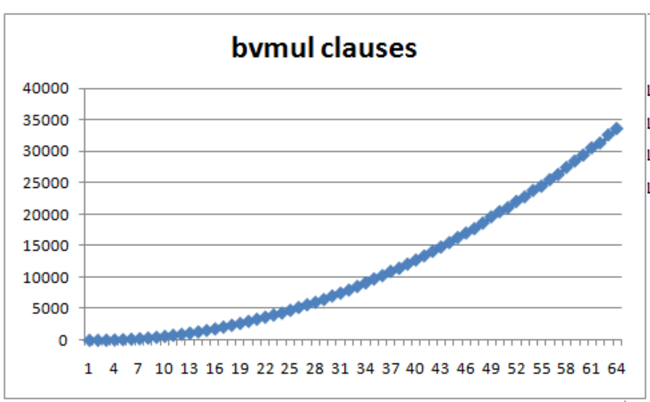
Bit-vector multiplication using SAT

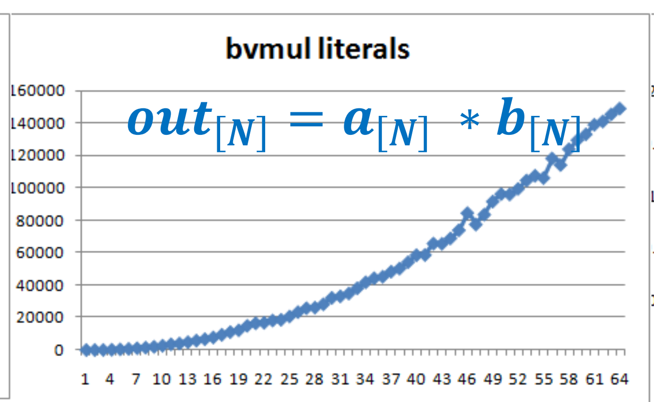


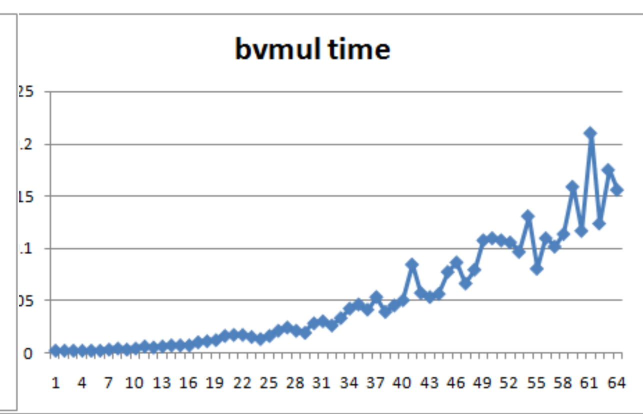
$O(n^2)$ clauses

SAT solving time increases exponentially. Similar for BDDs. [Bryant, MC25, 08]

Brute-force enumeration + evaluation faster for 20 bits. [Matthews, BPR 08]







A Framework and its limitations

- DPLL(T) is Z3's main core search framework

Efficient SAT technologies

- *DPLL + CDCL + Restart = Space Efficient Resolution*

Efficient integration of incremental theory solvers

- *Theory lemmas* *(T-Conflicts)*
- *Theory propagation* *(T-Propagation)*

But we claim

- *Contemporary DPLL(T) < Resolution*

A Framework and its limitations

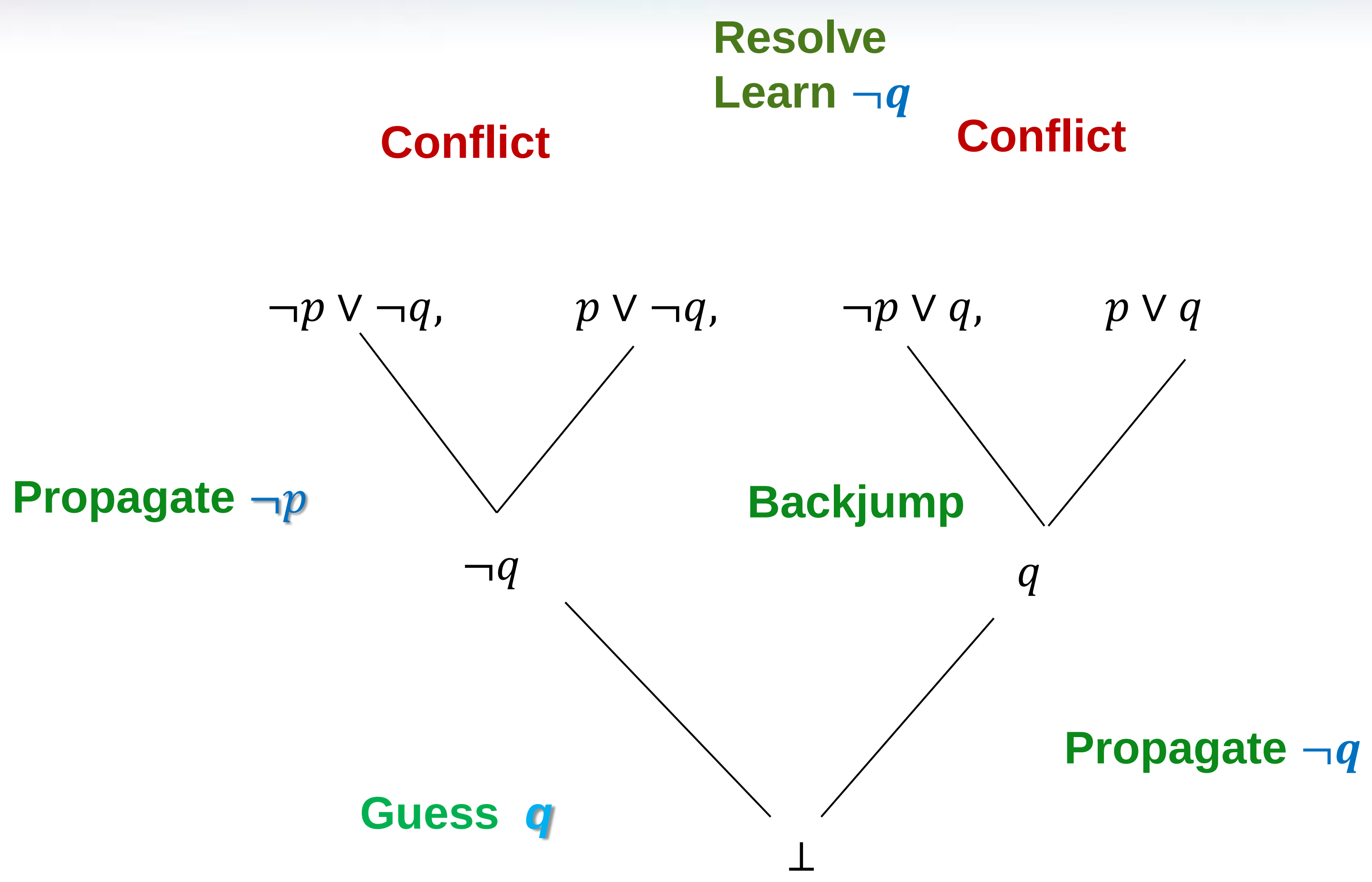
But … DPLL(T) < Resolution

Possible remedies:

- Forget DPLL(T). Use other core engine.
- Adapt DPLL(T). Elaboration here. We call it:

Conflict Directed Theory Resolution

Review: SAT made "tractable"



Review: SAT made "tractable"

- *Builds resolution proof*
  - *General Resolution ≡ DPLL + CDCL + Restart*

(CDCL: Conflict Directed Clause Learning)

- *Space Efficient*
  - *DPLL does not create intermediary clauses*

- *Efficient indexing and heuristics*
  - *2-watch literals, Restarts, phase selection, clause minimization*

Review: Modern DPLL in a nutshell

Initialize	$\epsilon \mid F$	F *is a set of clauses*
Decide	$M \mid F \implies M, \ell \mid F$	ℓ *is unassigned*
Propagate	$M \mid F, C \vee \ell \implies M, \ell^{C \vee \ell} \mid F, C \vee \ell$	C *is false under* M
Conflict	$M \mid F, C \implies M \mid F, C \mid C$	C *is false under* M
Resolve	$M \mid F \mid C' \vee \neg\ell \implies M \mid F \mid C' \vee C$	$\ell^{C \vee \ell} \in M$
Learn	$M \mid F \mid C \implies M \mid F, C \mid C$	
Backjump	$M \neg\ell M' \mid F \mid C \vee \ell \implies M \ell^{C \vee \ell} \mid F$	C *has no literals in* M'
Unsat	$M \mid F \mid \emptyset \implies Unsat$	
Sat	$M \mid F \implies M$	F *true under* M
Restart	$M \mid F \implies \epsilon \mid F$	

Adapted and modified from [Nieuwenhuis, Oliveras, Tinelli J.ACM 06]

DPLL(T) in a nutshell

T- Propagate $M \mid F, C \vee \ell \implies M, \ell^{C \vee \ell} \mid F, C \vee \ell$ $\quad C\ is\ false\ under\ T + M$

T- Conflict $M \mid F \implies M \mid F \mid \neg M'$ $\quad M' \subseteq M\ and\ M' is\ false\ under\ T$

T- Propagate $a > b, b > c \mid F, a \leq c \vee b \leq d \implies$

$$a > b, b > c, b \leq d^{a \leq c \vee b \leq d} \mid F, a \leq c \vee b \leq d$$

T- Conflict $M \mid F \implies M \mid F,\ a \leq b \vee b \leq c \vee c < a$

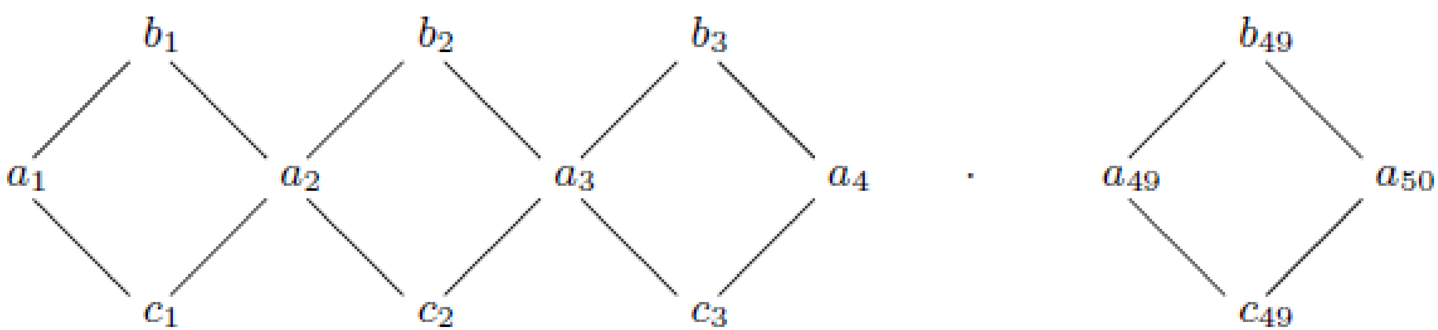
$$where\ a > b, b > c, a \leq c \subseteq M$$

Introduces no new literals - terminates

DPLL(T) misses short proofs

The **Black Diamonds** of DPLL(T)

$$\neg(a_1 \simeq a_{50}) \;\wedge\; \bigwedge_{i=1}^{49} [(a_i \simeq b_i \wedge b_i \simeq a_{i+1}) \vee (a_i \simeq c_i \wedge c_i \simeq a_{i+1})]$$



Has no short DPLL(T) proof.

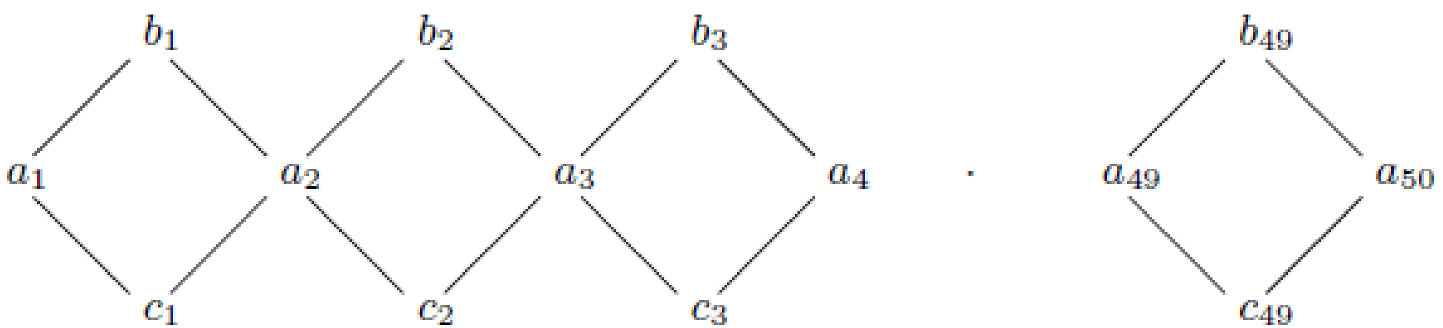
Has short DPLL(T) proof when using $a_1 \simeq a_2, a_2 \simeq a_3, a_3 \simeq a_4, \ldots, a_{49} \simeq a_{50}$

Example from [Rozanov, Strichman, SMT 07]

DPLL(T) misses short proofs

Idea: DPLL(⊔)

[B, Dutertre, de Moura 08]



Try branch $a_1 \simeq b_1 \wedge b_1 \simeq a_2$
Implies $a_1 \simeq b_1 \simeq a_2$
Collect implied equalities

Try branch $\neg(a_1 \simeq b_1 \wedge b_1 \simeq a_2)$
Implies $a_1 \simeq c_1 \simeq a_2$
Collect implied equalities

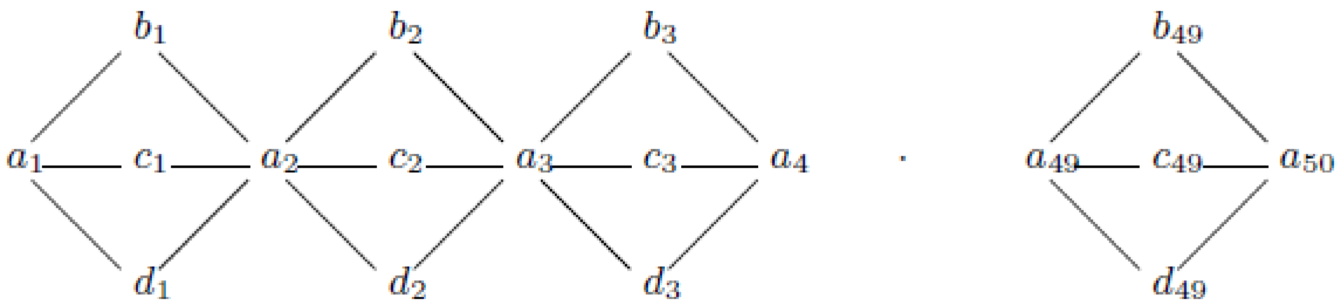
Compute the ***join*** ⊔ of the two equalities – common equalities are learned

Still potentially O(n^2) rounds just at ***base*** level of search.

DPLL(⊔ base) misses short proofs

- Single case splits don't suffice

$$a_1 \not\simeq a_{50} \;\wedge\; \bigwedge_{i=1}^{49} \left[\begin{array}{l} \quad (a_i \simeq b_i \wedge b_i \simeq a_{i+1}) \\ \vee\ (a_i \simeq c_i \wedge c_i \simeq a_{i+1}) \\ \vee\ (a_i \simeq d_i \wedge d_i \simeq a_{i+1}) \end{array} \right]$$



Requires 2 case splits to collect implied equalities

Conflict Directed Theory Resolution

We now describe an approach we call:

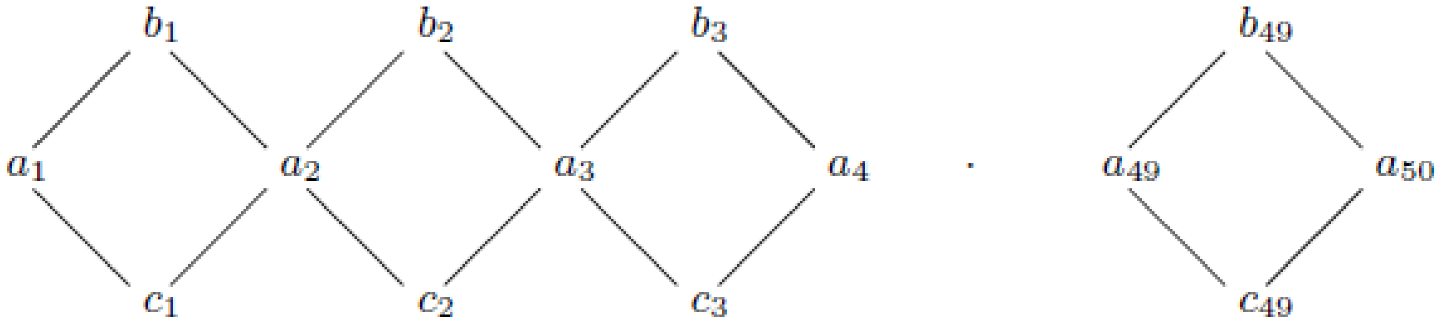
Conflict Directed Theory Resolution

resolve literals from conflicts
→ simulates resolution proofs.

Engineering: **Throttle** resolution dynamically based on activity.

Th(Equality) - Example

$$\neg(a_1 \simeq a_{50}) \wedge \bigwedge_{i=1}^{49} [(a_i \simeq b_i \wedge b_i \simeq a_{i+1}) \vee (a_i \simeq c_i \wedge c_i \simeq a_{i+1})]$$



Eventually, many conflicts contain: $a_1 \simeq b_1 \wedge b_1 \simeq a_2$

Use E-resolution, add clause: $a_1 \simeq b_1 \wedge b_1 \simeq a_2 \rightarrow a_1 \simeq a_2$

Then DPLL(T) learns by itself: $a_1 \simeq a_2$

Th(Equality) - Example

$$\bigwedge_{i=1}^{N}(p_i \vee x_i \simeq v_0) \wedge (\neg p_i \vee x_i \simeq v_1) \wedge (p_i \vee y_i \simeq v_0) \wedge (\neg p_i \vee y_i \simeq v_1) \wedge$$
$$\neg(f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots))$$

Eventually, many conflicts contain:

$$x_i \simeq u_i \wedge y_i \simeq u_i \quad u_i = v_0 \; or \; u_i = v_1 \quad for \; i = 1..N$$
$$\neg(f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots))$$

Add:

$$(\bigwedge_{i=1}^{N} x_i \simeq y_i) \rightarrow f(x_N, \ldots, f(x_2, x_1) \ldots) \simeq f(y_N, \ldots, f(y_2, y_1) \ldots)$$

Deciding Th(Equality)

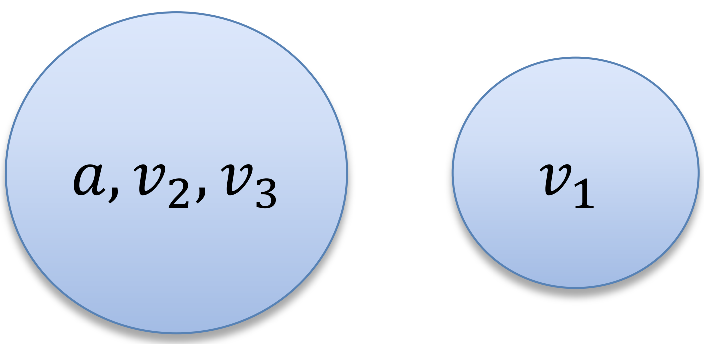
$$a = f(f(a)),\ a = f(f(f(a))),\ a \neq f(a)$$

First Step: “Naming” subterms

Deciding Th(Equality)

$$a = v_2, a = v_3, a \neq v_1,$$
$$v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2)$$

… and merge equalities

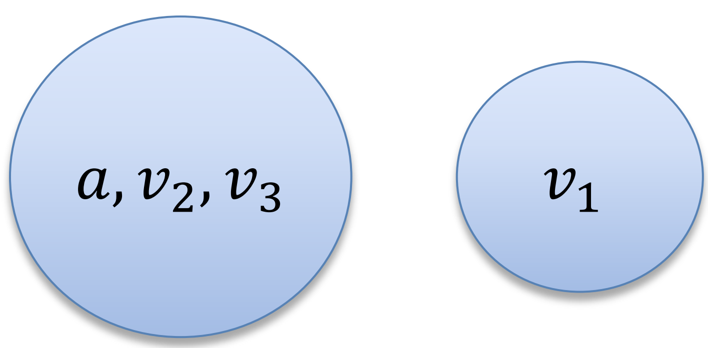


Deciding Th(Equality)

$$a = v_2, a = v_3, a \neq v_1,$$
$$v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2)$$

Second step. Apply Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$



Deciding Th(Equality)

$$a = v_2, a = v_3, a \neq v_1,$$
$$v_1 \equiv f(a), v_2 \equiv f(v_1), v_3 \equiv f(v_2)$$

Second step. Apply Congruence Rule:

$a \simeq v_2$ implies $f(a) \simeq f(v_2)$: $v_1 \simeq v_3$

a, v_2, v_3, v_1

CDTR for Th(Equalities)

Dynamic Ackermann Reduction

If *Congruence Rule* repeatedly learns

$$f(v, v') \sim f(w, w')$$

Then add clause for SAT core to use

$$v \simeq w \wedge v' \simeq w' \rightarrow f(v, v') \simeq f(w, w')$$

Used in Yices and Z3 to find short congruence closure proofs
[Yices Tool 06, Dutertre, de Moura]
[Model-based Theory Combination 07, de Moura, B]

CDTR for Th(Equalities)

Dynamic Ackermann Reduction

If *Congruence Rule* repeatedly learns

$$f(v, v') \sim f(w, w') \text{ for literal } f(v, v') \simeq f(w, w')$$

Then add clause for SAT core to use

$$v \simeq w \wedge v' \simeq w' \rightarrow f(v, v') \simeq f(w, w')$$

Leo identified the following useful optimization filter heuristic used in Z3

"Peel the onion from outside"

CDTR for Th(Equalities)

Dynamic Ackermann Reduction

If *Congruence Rule* repeatedly learns

$$f(v, v') \sim f(w, w')$$

Then add clause for SAT core to use

$$v \simeq w \wedge v' \simeq w' \rightarrow f(v, v') \simeq f(w, w')$$

Dynamic Ackermann Reduction with Transitivity

If *Equality Transitivity* repeatedly learns

$$u \sim w \qquad\qquad from\, u \sim v \, and \, v \sim w$$

Then add clause for SAT core to use

$$u \simeq v \wedge v \simeq w \rightarrow v \ \simeq w$$

CDTR: Th(Equalities)

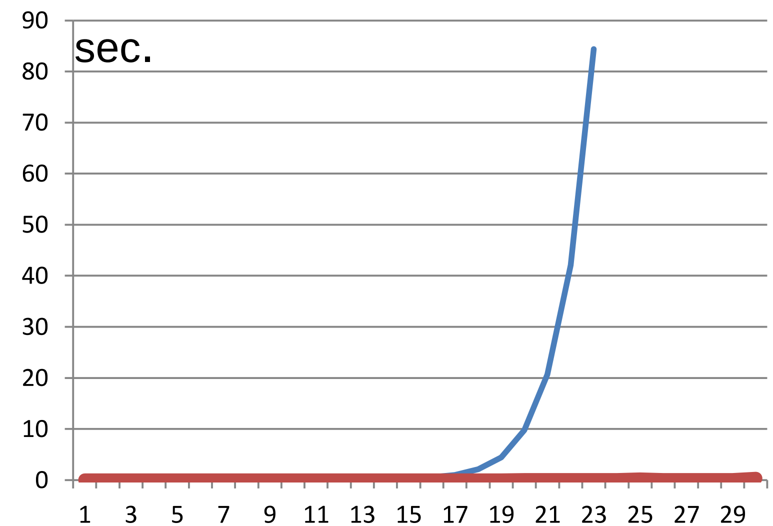
Claim:

Ground E-Resolution

≡

DPLL(E) + Dynamic Ackermann Reduction with Transitivity

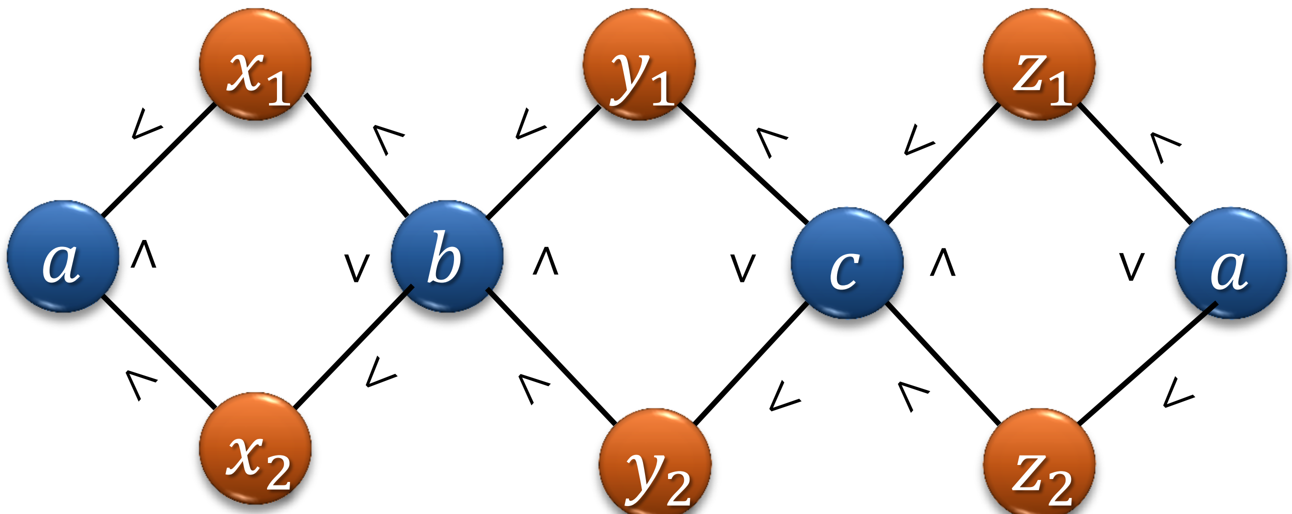
Effect on the Diamond Example:.



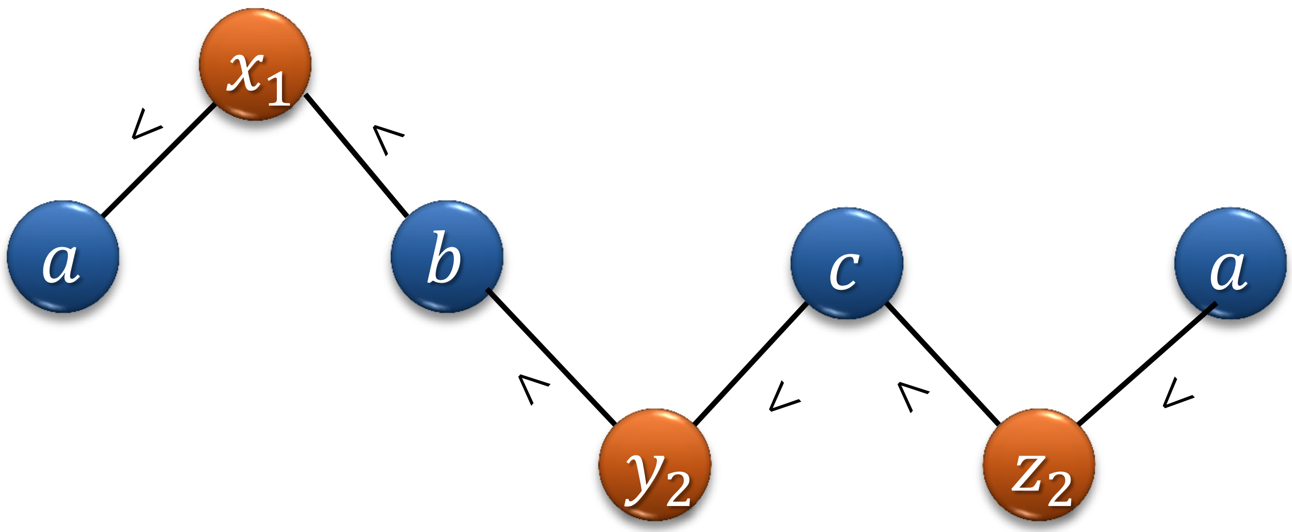
Alternative: Static Ackermann Reduction [Singerman, Pnueli, Velev, Bryant, Strichman, Lahiri, Seisha, Bruttomesso,Cimatti, Franzen, Griggio, Santuari, Sebastiani]
P-simulates ground E-Resolution.
But it has high up-front space overhead.

CDTR for *Linear Difference Arithmetic*

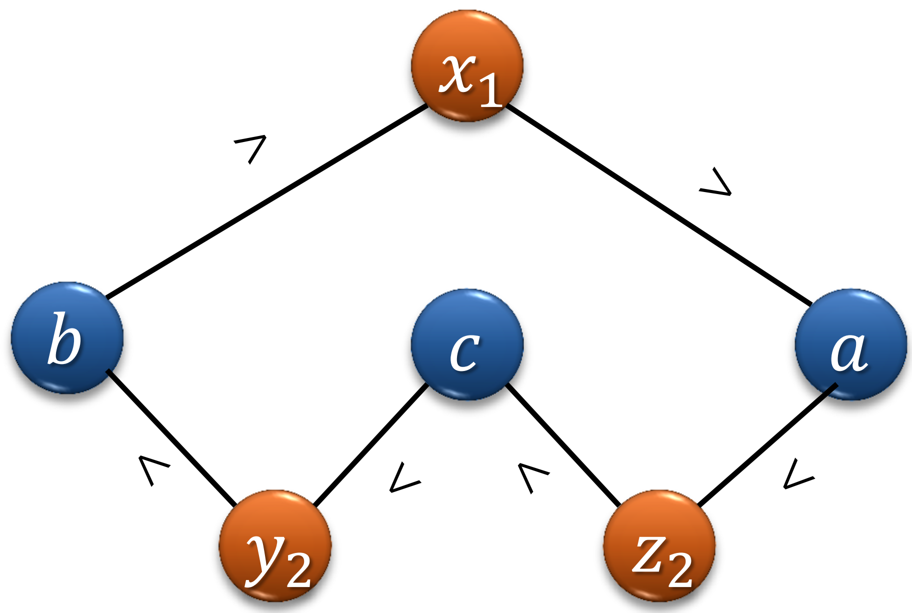
$$a < x_1 \wedge a < x_2 \wedge (x_1 < b \vee x_2 < b) \wedge$$
$$b < y_1 \wedge b < y_2 \wedge (y_1 < c \vee y_2 < c) \wedge$$
$$c < z_1 \wedge c < z_2 \wedge (z_1 < a \vee z_2 < a)$$



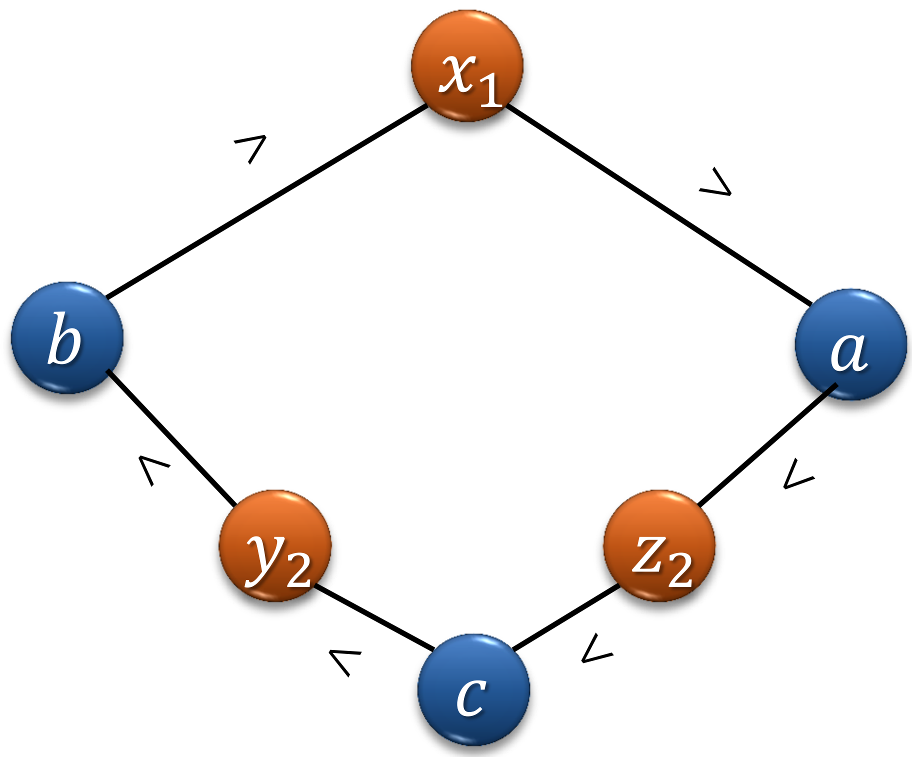
CDTR: *Linear Difference Arithmetic*



CDTR: *Linear Difference Arithmetic*

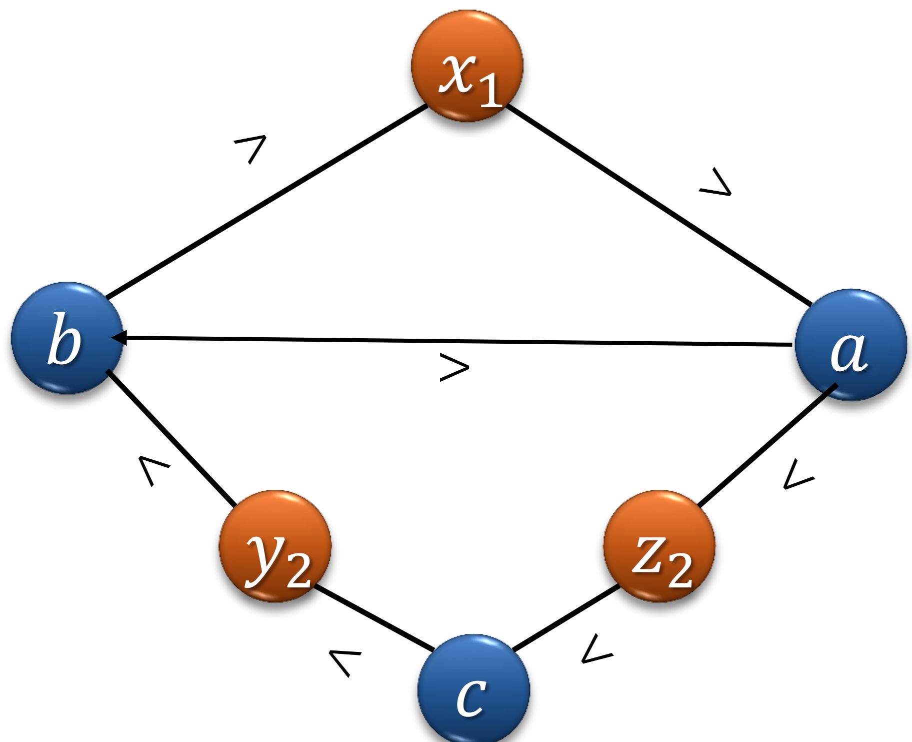


CDTR: *Linear Difference Arithmetic*



CDTR: *Linear Difference Arithmetic*

Top Two Most Active vertices



Add clause

$a < x_1 < b \rightarrow a < b$

Context and Extensions

Z3 supported theories all reduce to one of

Arithmetic	Equality	Booleans

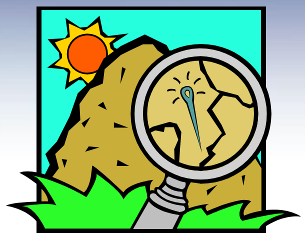
CDTR

Th(Equalities):	Extended Dynamic Ackermann
Th(Differences):	Cutting loops
Th(LRA):	Fourier-Motzkin resolution
Th(LIA):	Perhaps: Integer FM [B. IJCAR 10]

CDTR and theory combinations:

- Theories communicate equalities between shared variables.
- Build clauses using these equalities.

Summary



- Modern SMT solvers are tuned to but limitations of basic proof calculus shows up.
- Presented a technique to close the gap
  - **Dynamic** - to make it practical.
  - Based on applying **Resolution** to conflicts.
- Just one of many possible optimizations.
  - The quest for improving search continues
  - e.g. cutting plane proofs, arbitrary cuts (Frege)