Satisfiability modulo theory

David Monniaux

CNRS / VERIMAG

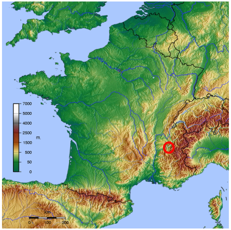
October 2017

Schedule

Who I am

David Monniaux
Senior researcher (*directeur de recherche*) at **CNRS** (National center for scientific research)

Working at **VERIMAG** in Grenoble



Who we are

VERIMAG = joint research unit of

- CNRS (9 permanent researchers)
- Université Grenoble Alpes (15 faculty)
- Grenoble-INP (8 faculty)

What we do

(Widely speaking)
Methods for designing and verifying safety-critical embedded systems

- formal methods
  - static analysis
  - proof assistants
  - decision procedures
  - exact analysis
- modeling of hardware/software system platforms (caches, networks on chip etc.)
- hybrid systems

Program verification

Program = instructions in

- "toy" programming language or formalism (Turing machines, λ-calculus, "while language")
- real programming language

Program has a set of behaviors (**semantics**)

Prove these behaviors included in acceptable behaviors (**specification**)

Difficulties

Defining the semantics of a real language (C, C++...) is **very difficult**.
Add parallelism etc. to make it nearly impossible.

Specifications are written by humans, may be themselves buggy.

I will concentrate on the **proving** phase.

Objectives

Increasing ambition:

Advanced testing find execution traces leading to violations
More efficiently than by hand or randomly

Assisted proof help the user prove the program

Automated proof prove the program automatically

Schedule

Schedule

Quantifier-free propositional logic

$\wedge$ (and), $\vee$ (or), $\neg$ (not) (also noted $\bar{x}$)
Possibly add $\oplus$ (exclusive-or)

t = "true", **f** = "false"

Formula with variables: $(a \vee b) \wedge (c \vee \bar{a})$

An **assignment** gives a value to all variables.
A **satisfying assignment** or **model** makes the formula evaluate to **t**.

Disjunctive normal form

Disjunction of conjunction of **literals** (= variable or negation of variable)

Obtained by distributivity
$(a \lor b) \land c \longrightarrow (a \land c) \lor (b \land c)$

Inconvenience?

Disjunctive normal form

Disjunction of conjunction of **literals** (= variable or negation of variable)

Obtained by distributivity
$(a \vee b) \wedge c \longrightarrow (a \wedge c) \vee (b \wedge c)$

Inconvenience?
Exponential size in the input.

Conjunctive normal form

Disjunction of literals = **clause**
CNF = conjunction of clauses

Obtained by distributivity

$$(a \wedge b) \vee c \longrightarrow (a \vee c) \wedge (b \vee c)$$

Negation normal form

Push negations to the leaves of the syntax tree using De Morgan's laws

$$\neg(a \vee b) \longrightarrow (\neg a) \wedge (\neg b)$$

$$\neg(a \wedge b) \longrightarrow (\neg a) \vee (\neg b)$$

Makes formula "monotone" with respect to $\mathbf{f} < \mathbf{t}$ ordering.

Applications

Verification of combinatorial circuits

Equivalence of two circuits

Schedule

The problem

Representing **compactly** set of Boolean states

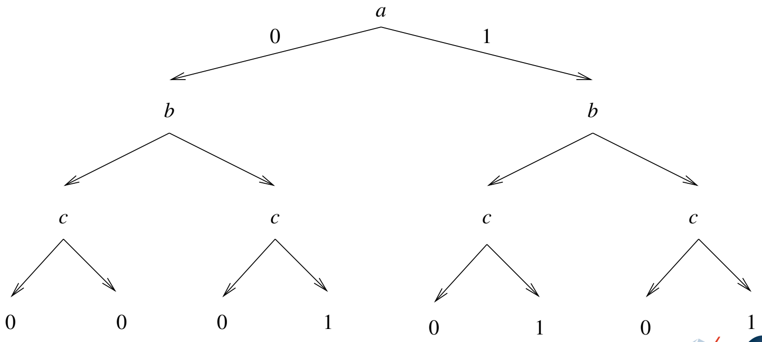
A set of vector *n* Booleans = a function from $\{0, 1\}^n$ into $\{0, 1\}$.

Example: $\{(0, 0, 0), (1, 1, 0)\}$ represented by $(0, 0, 0) \mapsto 1$, $(1, 1, 0) \mapsto 1$ and 0 elsewhere.

Expanded BDD

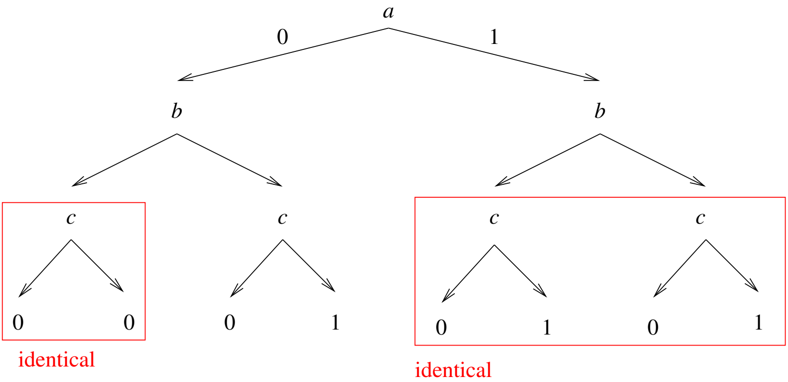
BDD = Binary decision diagram

Given ordered Boolean variables (a, b, c), represent $(a \wedge c) \vee (b \wedge c)$:

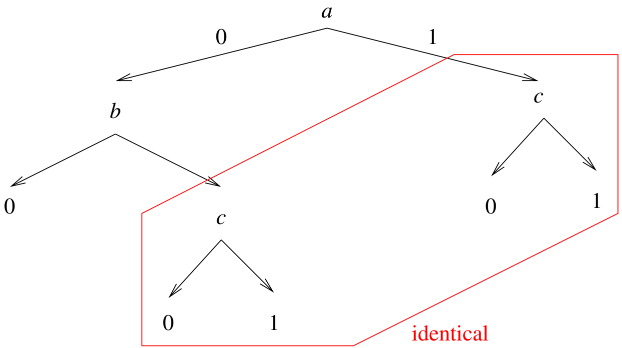


Removing useless nodes

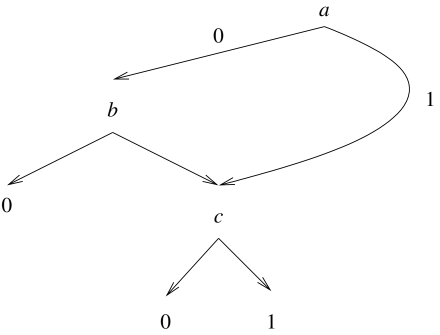
Silly to keep two identical subtrees:



Compression



Reduced BDD



Idea: turn the original tree into a DAG with **maximal sharing**.

Two different but isomorphic subtrees are never created.
Canonicity: a given example is always encoded by the same DAG.

Implementation: hash-consing

Important: implementation technique that you may use in other contexts

"Consing" from "constructor" (cf Lisp : `cons`).

> *In computer science, particularly in functional programming, hash consing is a technique used to share values that are structurally equal. [...] An interesting property of hash consing is that two structures can be tested for equality in constant time, which in turn can improve efficiency of divide and conquer algorithms when data sets contain overlapping blocks.*

`https://en.wikipedia.org/wiki/Hash_consing`

Implementation: hash-consing 2

Keep a **hash table** of all nodes created, with hashcode $H(x)$ computed quickly.
If node = (v, b_0, b_1) compute H from v and unique identifiers of b_0 and b_1

Unique identifier = address (if unmovable) or serial number

If an object matching (v, b_0, b_1) already exists in the table, return it

How to collect garbage nodes? (unreachable)

Garbage collection in hash consing

Needs **weak pointers**: the pointer from the hash table should be ignored by the GC when it computes reachable objects

- Java `WeakHashMap`
- OCaml `Weak`

Garbage collection in hash consing

Needs **weak pointers**: the pointer from the hash table should be ignored by the GC when it computes reachable objects

- Java `WeakHashMap`
- OCaml `Weak`

(Other use of weak pointers: caching recent computations.)

Hash-consing is magical

Ensures:

- **maximal sharing:** never two identical objects in two $\neq$ locations in memory
- ultra-fast equality test: sufficient to **compare pointers** (or unique identifiers)

And once we have it, BDDs are easy.

BDD operations

Once a variable ordering is chosen:

- Create BDD **f**, **t**(1-node constants).
- Create BDD for v, for v any variable.
- Operations $\wedge$, $\vee$, etc.

Binary BDD operations

Operations $\wedge$, $\vee$: recursive descent on both subtrees, with **memoizing**:

- store values of $f(a, b)$ already computed in a hash table
- index the table by the unique identifiers of a and b

Complexity with and without dynamic programming?

Binary BDD operations

Operations $\wedge$, $\vee$: recursive descent on both subtrees, with **memoizing**:

- store values of $f(a, b)$ already computed in a hash table
- index the table by the unique identifiers of a and b

Complexity with and without dynamic programming?

- without dynamic programming: unfolds DAG into tree $\Rightarrow$ exponential
- with dynamic programming $O(|a|.|b|)$ where $|x|$ the size of DAG x

Fixed point solving

B_0 = initial states
$B_1 = B_0$ + reachable in 1 step from B_0
$B_2 = B_1$ + reachable in 1 step from B_1
$B_3 = B_2$ + reachable in 1 step from B_2
$\vdots$
converges in finite time to R = **reachable states**

Iteration sequence

$$B_{k+1}(x'_1, \ldots, x'_n) =$$
$$\exists x_1 \ldots x_n \; B_k(x_1, \ldots, x_n) \wedge \tau(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$$

Needs

- variable renaming (easy)
- constructing the BDD for τ
- $\exists$-elimination for n variables
- conjunction

Tools

e.g. NuSMV, NuXMV in BDD mode

Schedule

SAT problem

Given a propositional formula say whether it is satisfiable or not (give a model if so)./
Satisfiable = "it has a model" = "it has a satisfying assignment".

- "SAT" with arbitrary formula
- "CNF-SAT" with formula in CNF
- "3CNF-SAT" with formula in CNF, 3 literals per clause

Exercise

Show that one can convert SAT into 3CNF-SAT with time and output linear in size of input.

Tseitin encoding

Gregory S. Tseytin, *On the complexity of derivation in propositional calculus*, `http://www.decision-procedures.org/handouts/Tseitin70.pdf`

Can one transform a formula into another in CNF with **linear** size and preserve solutions?

$$(a \vee b) \wedge (c \vee d) \longrightarrow (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d)$$

Tseitin encoding

Add extra variables

$$\left((a \wedge \bar{b} \wedge \bar{c}) \vee (b \wedge c \wedge \bar{d})\right) \wedge (\bar{b} \vee \bar{c})\,.$$

Assign propositional variables to sub-formulas:

$$\begin{array}{ccc} e \equiv a \wedge \bar{b} \wedge \bar{c} & f \equiv b \wedge c \wedge \bar{d} & g \equiv e \vee f \\ h \equiv \bar{b} \vee \bar{c} & \phi \equiv g \wedge h\,; & \end{array}$$

Tseitin encoding

$$e \equiv a \wedge \bar{b} \wedge \bar{c} \qquad f \equiv b \wedge c \wedge \bar{d} \qquad g \equiv e \vee f$$
$$h \equiv \bar{b} \vee \bar{c} \qquad \phi \equiv g \wedge h\,;$$

turned into clauses

$$\begin{array}{cccc}
\bar{e} \vee a & \bar{e} \vee \bar{b} & \bar{e} \vee \bar{c} & \bar{a} \vee b \vee c \vee e \\
\bar{f} \vee b & \bar{f} \vee c & \bar{f} \vee d & \bar{b} \vee \bar{c} \vee d \vee f \\
\bar{e} \vee g & \bar{f} \vee g & \bar{g} \vee e \vee f & \\
b \vee h & c \vee h & \bar{h} \vee \bar{b} \vee \bar{c} & \\
\bar{\phi} \vee g & \bar{\phi} \vee h & \bar{g} \vee \bar{h} \vee \phi & \phi
\end{array}$$

DIMACS format

```
c start with comments
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

5 = number of variables

3 = number of clauses

each clause: -1 is variable 1 negated, 5 is variable 5, 0 is end of clause

DPLL

Each clause acts as **propagator** e.g.
assuming a and $\bar{b}$, clause $\bar{a} \vee b \vee c$ yields c

Boolean constraint propagation aka **unit propagation**:
propagate as much as possible
once the value of a variable is known, use it elsewhere

	7	2	3	8	5	4		
	3	9		1	6			
1			2	7		3		6
7	8					6	4	
5								7
	9	4					3	1
4		1		6	3			8
			9	2		1	6	
		8	5	4	1	2	7	

DPLL: Branching

If unit propagation insufficient to

- either find a satisfying assignment
- either find an unsatisfiable clause (all literals forced to false)

Then:

- pick a variable
- do a search subtree for both polarities of the variable

Example

$$
\begin{array}{cccc}
\bar{e} \vee a & \bar{e} \vee \bar{b} & \bar{e} \vee \bar{c} & \bar{a} \vee b \vee c \vee e \\
\bar{f} \vee b & \bar{f} \vee c & \bar{f} \vee d & \bar{b} \vee \bar{c} \vee d \vee f \\
\bar{e} \vee g & \bar{f} \vee g & \bar{g} \vee e \vee f & \\
b \vee h & c \vee h & \bar{h} \vee \bar{b} \vee \bar{c} & \\
\bar{\phi} \vee g & \bar{\phi} \vee h & \bar{g} \vee \bar{h} \vee \phi & \phi
\end{array}
$$

From unit clause ϕ

$\bar{\phi} \vee g \rightarrow g$ $\quad$ $\bar{\phi} \vee h \rightarrow h$ $\quad$ $\bar{g} \vee \bar{h} \vee \phi$ removed

Now g and h are **t**,

$\bar{e} \vee g$ removed $\quad$ $\bar{f} \vee g$ removed $\quad$ $b \vee h$ removed

$c \vee h$ removed $\quad$ $\bar{g} \vee e \vee f \rightarrow e \vee f$ $\quad$ $\bar{h} \vee \bar{b} \vee \bar{c} \rightarrow \bar{b} \vee \bar{c}$

CDCL: clause learning

A DPLL branch gets closed by **contradiction**: a literal gets forced to both **t** and **f**.

Both **t** and **f** inferred from hypotheses H by unit propagation. Trace back to a subset of hypotheses, sufficient for contradiction.

e.g. $a \wedge \bar{b} \wedge \bar{c} \wedge d \wedge H \implies \mathbf{f}$

Learn clause = negation of bad hypotheses, implied by H:

$$\bar{a} \vee b \vee c \vee \bar{d}$$

Add this clause (maybe garbage-collected later) to H
Used by unit propagation

Resolution

Proved C_1 from $H_1, \ldots, H_n$ and hypothesis a +
proved C_2 from $H_1, \ldots, H_n$ and hypothesis $\bar{a}$
$\equiv$
Proved $C_1 \vee \bar{a}$ from $H_1, \ldots, H_n$ +
proved $C_2 \vee a$ from $H_1, \ldots, H_n$

Propositional resolution rule:

$$\dfrac{\begin{array}{c} H_1 \quad \ldots \quad H_n \\ \vdots \\ C_1 \vee \bar{a} \end{array} \qquad \begin{array}{c} H_1 \quad \ldots \quad H_n \\ \vdots \\ C_2 \vee a \end{array}}{C}$$

Resolution proofs

a DPLL “unsat” run = a resolution **tree** proof
obtain it by instrumenting the solver =
logging the proof steps used for deriving clauses

a CDCL “unsat” run = a resolution **DAG** proof
shared subtrees for learned lemmas

CDCL better than DPLL

Some problems:

- short DAG resolution proofs
- only exponential tree resolution proofs.

Resolution independent of search strategy!

Almost all current solvers do CDCL, not DPLL.

Insights

Difficult to prove results on solvers full of heuristics

Can sometimes prove properties of their **proof systems**

Pigeons

n pigeons, m pigeon holes
$a_{i,j}j$ means pigeon i in hole j
Each pigeon in a hole: for all i,

$$a_{i,1} \vee \cdots \vee a_{1,m}$$

Each hole has at most one pigeon: for all j, for all i, i',

$$\bar{a}_{i,j} \vee \cdots \vee \bar{a}_{i',j}$$

If $n > m$ "trivially" unsatisfiable
...but any DAG resolution proof has exponential size in $n = m + 1$

Implementation wise

Clause simplification etc. implemented as
two watched literals per clause

Pointers to clauses used for deduction

Highly optimized proof engines

- Minisat `http://minisat.se/`
- Glucose `http://www.labri.fr/perso/lsimon/glucose/`
- Armin Biere's solvers `http://fmv.jku.at/software/index.html`

Preprocessing

Unit propagation

Textbook propagation

- look for a variable in each clause, “remove it”
- how do we backtrack?

Remark: a clause acts when 1 unassigned literal (and none assigned $\mathbf{t}$)

$$a \vee \bar{b} \vee c \vee \bar{d}$$

in context $a = \mathbf{f}$, $b = \mathbf{t}$, $d = \mathbf{t}$, deduce $c = \mathbf{t}$.

Two watched literals per clause

- for each literal: a linked list of clauses
- each clause has two watched literals
- invariant: in each clause the two watched literals are not assigned to false

When $l := \mathbf{f}$, scan associated list

- For each clause with one literal assigned $\mathbf{t}$, ignore the clause
- For each clause with > 1 unassigned literal l', move clause to the list for l'
- For each clause with 1 unassigned literal l', $l' := \mathbf{t}$
- 0 unassigned literal, CONFLICT (analyze and backtrack)

Variable and polarity selection

Heuristics for picking variable to branch on

Polarity (**t** vs **f**)

- heuristics for picking first polarity choice
- keep last polarity used in next choices

Restart once in a while (keep polarities and learned clauses)

Schedule

Equivalence checking

Two combinatorial circuits
How to check they compute the same?

Equivalence checking

Two combinatorial circuits
How to check they compute the same?
$(a_1, \dots, a_n)$ output from A
$(b_1, \dots, b_n)$ output from B (same inputs)
assert $(a_1 \oplus b_1) \vee \dots \vee (a_n \oplus b_n)$

Schedule

Schedule

Quantifier-free first order formulas

e.g. $(P(f(x, z), y) \wedge R(y) \wedge T) \vee S(y, g(z, x))$

Ordinary connectives $\wedge$, $\vee$, $\neg$...

Predicate symbols $P, Q, \ldots$, each with an **arity** (number of arguments)

- 0-ary predicates = propositional variables
- 1-ary predicates (monadic)
- other predicates

Function symbols $f, g, \ldots$, each with an **arity**
0-ary function symbols are known as **constants**

Common notations

Predicate and function symbols form a **signature**

Some predicate symbols may be noted as infix: $<$, $\leq$, $>$, $\geq$...
$=$ denotes **equality** (more on that later)

Some function symbols may be noted as infix: $+$, $-$
Some constants may be noted as 0, 1 etc.
1 may a **notation** for $S(0)$, 2 as $S(S(0))$ where S is successor

These so far implies **nothing** regarding the interpretation of these symbols.

Quantifiers

$\forall$, $\exists$

Can be moved to the front = **prenex** form.
Beware of variable capture!

$$\forall x\ ((\forall y P(x,y)) \lor (\exists y Q(x,y)))$$

in prenex form

$$\forall x \forall y_1 \exists y_2\ (P(x,y_1) \lor Q(x,y_2))$$

Interpretation

Let $\mathcal{M}$ be a set.

To any predicate symbol P of arity n associate $P^{\mathcal{M}} \subseteq \mathcal{M}^n$
Note: a 0-ary predicate associates to **t** or **f**

To any function symbol f of arity n associate $P^{\mathcal{M}} : \mathcal{M}^n \to \mathcal{M}$

To any **term** t (e.g. $g(z, f(x, y))$) associate an **interpretation** $t^{\mathcal{M}}$.
To any formula F associate an **interpretation** $F^{\mathcal{M}}$.

Interpretation with equality

In most cases:
The predicate $=$ must be interpreted as equality.

Models of a set of formulas

Let $\mathcal{F}$ be a set of formulas (or "system of axioms").

A **model** $\mathcal{M}$ is an interpretation that makes true all formulas in $\mathcal{F}$.

Schedule

Models with fixed interpretation of certain predicates

$$\forall x \exists y \; y > x \wedge y < x + z$$

E.g. $\mathcal{M}$ is the set $\mathbb{Z}$, $+$ is addition, $-$ subtraction, 0 is zero, S is "successor" $x \mapsto x + 1$.

E.g. $\mathcal{M}$ is the set $\mathbb{Q}$, $+$ is addition, $-$ subtraction, 0 is zero, S is $x \mapsto x + 1$.

The remaining part of the model, to fix, is then z.

Example

```
(set-option :produce-models true)
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(assert (<= x y))
(assert (<= y z))
(check-sat)
(get-model)
(push 1)
(assert (< z x))
(check-sat)
(pop 1)
(assert (<= z x))
(check-sat)
(get-model)
```

Output

```
cvc4 --incremental example.smt2
```

```
sat
(model
(define-fun x () Int 0)
(define-fun y () Int 0)
(define-fun z () Int 0)
)
unsat
sat
(model
(define-fun x () Int 0)
(define-fun y () Int 0)
(define-fun z () Int 0)
)
```

Schedule

DPLL(T)

(Improper terminology, should be CDCL(T))

$(x \leq 0 \vee x + y \leq 0) \wedge y \geq 1 \wedge x \geq 1$
$\downarrow$ dictionary of theory literals
$(a \vee b) \wedge c \wedge d$

Solve, get $(a, b, c, d) = (\mathbf{t}, \mathbf{f}, \mathbf{t}, \mathbf{t})$.
But $x \leq 0 \wedge x \geq 1$ is a contradiction!
Add **theory lemma** $\bar{a} \vee \bar{d}$

Solve, get $(a, b, c, d) = (\mathbf{f}, \mathbf{t}, \mathbf{t}, \mathbf{t})$.
But $x + y \leq 0 \wedge \geq 1 \wedge x \geq 1$ is a contradiction!
Add **theory lemma** $\bar{b} \vee \bar{c} \vee \bar{d}$.

The problem is **unsatisfiable**.

DPLL(T)

In practice, do not wait for the CDCL solver to provide a full assignment.
Check partial assignments for theory feasibility.

If during theory processing, a literal becomes known to be **t** or **f**, propagate it to CDCL.
e.g. $x \geq 0$, $x \geq 1$ assigned, propagate $x + y \geq 0$

Boolean relaxation of the original problem.
Lazy expansion of theory.

Linear real arithmetic

Usually decided by exact precision **simplex**.
Extract from the tableau the contradictory subset of assignments.

LRA Example

$$\left\{\begin{array}{rll} 2 & \leq 2x + y & \\ -6 & \leq 2x - 3y & \\ -1000 & \leq 2x + 3y & \leq 18 \\ -2 & \leq -2x + 5y & \\ 20 & \leq x + y\,. & \end{array}\right. \tag{1}$$

LRA Example

$$\begin{cases} a = & 2x & +y & 2 & \leq a & \\ b = & 2x & -3y & -6 & \leq b & \\ c = & 2x & 3y & -1000 & \leq c & \leq 18 \\ d = & -2x & +5y & -2 & \leq d & \\ e = & x & +y & 20 & \leq e\,. & \end{cases} \tag{2}$$

LRA Example

Gauss-like pivoting until:

$$\left\{\begin{array}{rrr} e = & 7/16c & -1/16d \\ a = & 3/4c & -1/4d \\ b = & 1/4c & -3/4d \\ x = & 5/16c & -3/16d \\ y = & 1/8c & +1/8d\,. \end{array}\right. \tag{3}$$

LRA Example

$e = 7/16c - 1/16d$
But: $c \leq 18$ and $d \geq -2$, so $-7/16c - 1/16d \leq 8$.
But we have $e \geq 20$, thus **no solution**.

Relevant original inequalities can be combined into an unsatisfiable one (thus the **theory lemma**)

$$\begin{array}{llrcrl} 7/16 & (-2x & -3y) & \geq & -7/16 & \times 18 \\ 1/16 & (-2x & +5y) & \geq & -1/16 & \times 2 \\ 1 & x & +y & \geq & 20 & \\ \hline & 0 & 0 & \geq & 12 & \end{array} \tag{4}$$

Linear integer arithmetic

Linear real arithmetic +

- branching: if LRA model $x = 4.3$, then $x \leq 4 \vee x \geq 5$
- (sometimes) Gomory cuts

Uninterpreted functions

$f(x) \neq f(y) \land x = z + 1 \land z = y - 1$
$\downarrow$
$f_x \neq f_y \land x = z + 1 \land z = y - 1$

Get $(x, y, z, f_x, f_y) = (1, 1, 0, 0, 1)$.
But if $x = y$ then $f_x = f_y$! Add $x = y \implies f_x = f_y$.

The problem over (x, y, z, f_x, f_y) becomes **unsatisfiable**.

Arrays

$update(f, x_0, y_0)$ the function mapping

- $x \neq x_0$ to $f[x]$
- x_0 to y_0.

Quantifiers

Show this formula is true:

$$(\forall i\, 0 \le i < j \implies t[i] = 42) \implies \\ (\forall i\, 0 \le i \le j \implies \mathit{update}(t, j, 0)[i] = 42) \quad (5)$$

Equivalently, unsatisfiable:

$$0 \le i_0 \le j \wedge \mathit{update}(t, j, 0)[i_0] = 0 \wedge (\forall i\, 0 \le i < j \implies t[i] = 0)$$

Instantiation

Prove unsatisfiable:

$$0 \leq i_0 \leq j \wedge update(t, j, 0)[i_0] = 0 \wedge (\forall i\, 0 \leq i < j \implies t[i] = 0)$$

By **instantiation** $i = i_0$:

$$0 \leq i_0 \leq j \wedge update(t, j, 0)[i_0] = 0 \wedge (0 \leq i_0 < j \implies t[i_0] = 0)$$

Unsatisfiable

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants

Inductiveness checking

Floyd-Hoare proof methods

Prove a property holds at every loop iteration:

- prove it holds initially
- prove: if it holds then it holds at next iteration

Proving $A \implies B$ universally $\equiv$
proving $A \land \neg B$ unsatisfiable

Example: binary search

```
/*@ requires
  @   n >= 0 && \valid(t+(0..n-1)) &&
  @   \forall int k1, k2; 0 <= k1 <= k2 <= n-1 ==> t[k1] <= t[k2];
  @ assigns \nothing;
  @ ensures
  @   (0 <= \result < n && t[\result] == v) ||
  @   (\result == -1 && \forall int k; 0 <= k < n ==> t[k] != v);
  @*/
int binary_search(int* t, int n, int v) {
  int l = 0, u = n-1;
  /*@ loop invariant
    @   0 <= l && u <= n-1
    @   && (\forall int k; 0 <= k < n ==> t[k] == v ==> l <= k <= u) ;
    @ loop assigns l,u ;
    @ loop variant u-l ;
    @*/
  while (l <= u ) {
    int m = l + (u-l) / 2;
    //@ assert l <= m <= u;
    if (t[m] < v) l = m + 1;
    else if (t[m] > v) u = m - 1;
    else return m;
  }
  return -1;
```

Symbolic / concolic execution

Explore the program:

- Follow **paths** inside the program
- On each path collect **constraints** on variables (guards in tests)
- Check feasibility using SMT-solving
- If symbolic execution becomes impossible (calls to native code...), **concretize** (find actual values) for some variables

Example

```c
#include <klee/klee.h>
int get_sign(int x) {
  if (x == 0)
     return 0;

  if (x < 0)
     return -1;
  else
     return 1;
}
int main() {
  int a;
  klee_make_symbolic(&a, sizeof(a), "a");
  return get_sign(a);
}
```

Running Klee

```
$ clang-3.4 -I $KLEE/include -emit-llvm -c -g g
$KLEE/bin/klee get_sign.bc

KLEE: output directory is "klee-out-0"
KLEE: Using STP solver backend

KLEE: done: total instructions = 31
KLEE: done: completed paths = 3
KLEE: done: generated tests = 3
```

Examining one test input

```
$ ktest-tool klee-last/test000002.ktest
ktest file : 'klee-last/test000002.ktest'
args       : ['get_sign.bc']
num objects: 1
object    0: name: 'a'
object    0: size: 4
object    0: data: '\x01\x01\x01\x01'
```

Checking an assertion failure

```c
#include <klee/klee.h>
int main() {
  int t = 0, x;
  for(int i=0; i<3; i++) {
    klee_make_symbolic(&x, sizeof(x), "xboucle"
    klee_assume((x >= 0) & (x < 100));
    t += x;
  }
  klee_assert(t < 290);
  return 0;
}
```

Crash trace

```
ktest file : 'klee-last/test000001.ktest'
args       : ['klee_boucle.bc']
num objects: 3
object    0: name: 'xboucle'
object    0: size: 4
object    0: data: 98
object    1: name: 'xboucle'
object    1: size: 4
object    1: data: 93
object    2: name: 'xboucle'
object    2: size: 4
object    2: data: 99
```

Bounded model checking

Convert a loop-free program into one big formula
One Boolean per control location = "the execution went through it"

Close to SSA form in compilers.
(Can be extended to arrays, structures, objects, pointers, pointer arithmetic. Becomes messy.)

If loops, unroll them to finite depth

BMC example

```c
extern int choice(void);

int main() {
  int t = 0, x;
  for(int i=0; i<3; i++) {
    x = choice();
    if (x > 100 || x < 0) x=0;
    t += x;
  }
  assert(t < 290);
  return 0;
}
```

BMC results

```
$ cbmc --trace cbmc_boucle.c
```

```
State 18 file cbmc_boucle.c line 4 function main thread 0
  t=0 (00000000000000000000000000000000)
State 19 file cbmc_boucle.c line 4 function main thread 0
  t=0 (00000000000000000000000000000000)
State 20 file cbmc_boucle.c line 4 function main thread 0
  x=0 (00000000000000000000000000000000)
State 21 file cbmc_boucle.c line 5 function main thread 0
  i=0 (00000000000000000000000000000000)
State 22 file cbmc_boucle.c line 5 function main thread 0
  i=0 (00000000000000000000000000000000)
State 27 file cbmc_boucle.c line 6 function main thread 0
  x=95 (00000000000000000000000001011111)
State 29 file cbmc_boucle.c line 8 function main thread 0
  t=95 (00000000000000000000000001011111)
State 30 file cbmc_boucle.c line 5 function main thread 0
  i=1 (00000000000000000000000000000001)
State 36 file cbmc_boucle.c line 6 function main thread 0
  x=97 (00000000000000000000000001100001)
State 38 file cbmc_boucle.c line 8 function main thread 0
  t=192 (00000000000000000000000011000000)
State 39 file cbmc_boucle.c line 5 function main thread 0
  i=2 (00000000000000000000000000000010)
State 45 file cbmc_boucle.c line 6 function main thread 0
  x=98 (00000000000000000000000001100010)
State 47 file cbmc_boucle.c line 8 function main thread 0
  t=290 (00000000000000000000000100100010)
State 48 file cbmc_boucle.c line 5 function main thread 0
  i=3 (00000000000000000000000000000011)

Violated property:
  file cbmc_boucle.c line 10 function main
  assertion t < 290
```

Schedule

Schedule

Motivating example

As in bounded model checking: Formula extracted from loop-free **software** (e.g. step function of a fly-by-wire controller):

- one Boolean per program basic block "the execution goes through that block"
- constraints expressing program operations and tests (e.g. instruction `x = x+1;` translated to $x_2 = x_1 + 1$

Solution of the formula $\equiv$ execution trace with all intermediate value

WCET

Worst-case execution time = time for the longest execution of the program

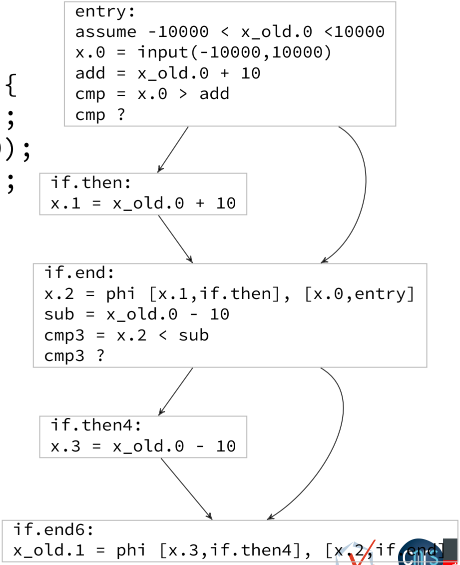
Enrich the formula with timing information for basic blocks
(In real life, this is more complicated)

Maximize the solution

SMT Encoding by Example

```c
void rate_limiter_step() {
  assume (x_old <= 10000);
  assume (x_old >= -10000);
  x = input(-10000,10000);
  if (x > x_old+10)
    x = x_old+10;
  if (x < x_old-10)
    x = x_old-10;
  x_old = x;
}

void main() {
  while (1)
    rate_limiter_step();
}
```



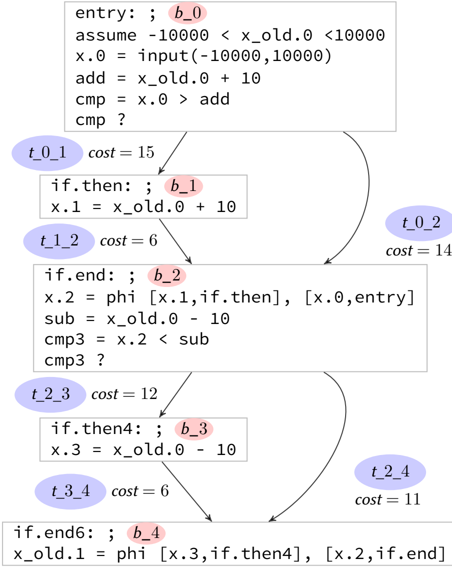
LLVM Control Flow Graph

The SMT formula encodes the feasible program traces:

- 1 Boolean per block
- 1 Boolean per transition

b_i *true* $\leftrightarrow$ trace goes through b_i

Cost for the trace:
$\sum b_i * cost_i$

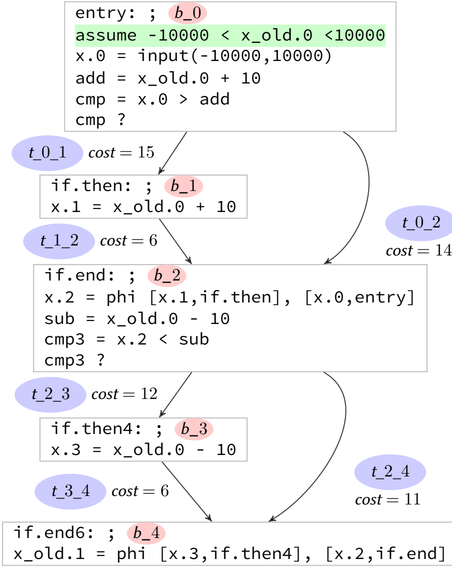


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

1 SMT variable ↔ 1 SSA variable

$$\begin{array}{ll} & -10000 \leq x\_old.0 \leq 10000 \\ \wedge & -10000 \leq x.0 \leq 10000 \\ \wedge & add = (x\_old.0 + 10) \\ \wedge & x.1 = (x\_old.0 + 10) \\ \wedge & sub = (x\_old.0 - 10) \\ \wedge & x.3 = (x\_old.0 - 10) \\ \wedge & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\ \wedge & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2)) \end{array}$$

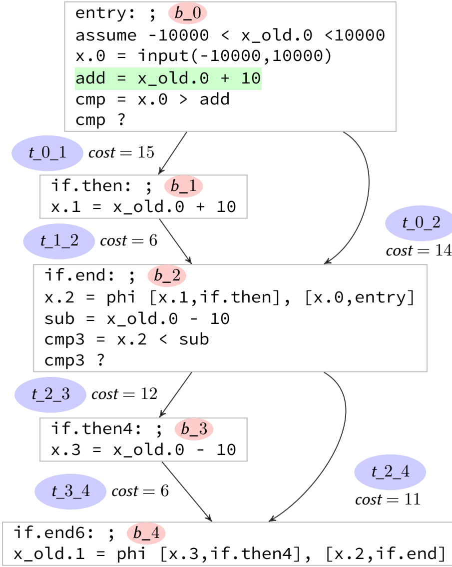


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

1 SMT variable $\leftrightarrow$ 1 SSA variable

$$
\begin{array}{ll}
 & -10000 \leq x\_old.0 \leq 10000 \\
\wedge & -10000 \leq x.0 \leq 10000 \\
\wedge & add = (x\_old.0 + 10) \\
\wedge & x.1 = (x\_old.0 + 10) \\
\wedge & sub = (x\_old.0 - 10) \\
\wedge & x.3 = (x\_old.0 - 10) \\
\wedge & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\
\wedge & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2))
\end{array}
$$

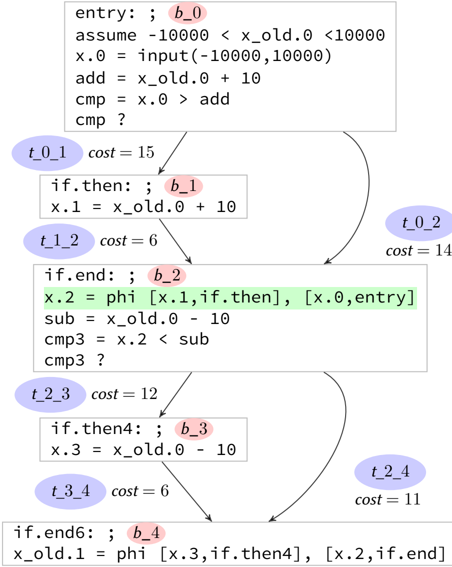


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

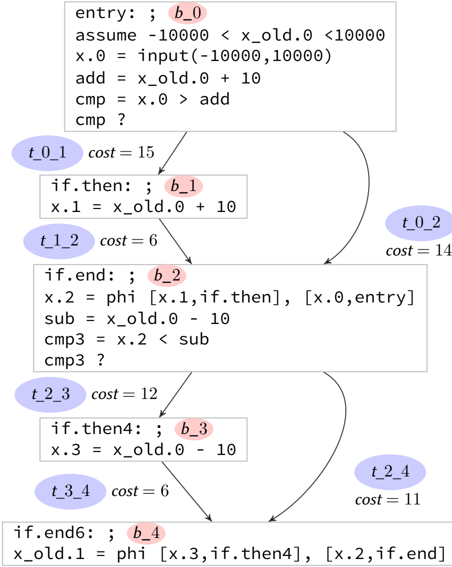
1 SMT variable $\leftrightarrow$ 1 SSA variable

$$
\begin{array}{ll}
 & -10000 \leq x\_old.0 \leq 10000 \\
\wedge & -10000 \leq x.0 \leq 10000 \\
\wedge & add = (x\_old.0 + 10) \\
\wedge & x.1 = (x\_old.0 + 10) \\
\wedge & sub = (x\_old.0 - 10) \\
\wedge & x.3 = (x\_old.0 - 10) \\
\wedge & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\
\wedge & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2))
\end{array}
$$



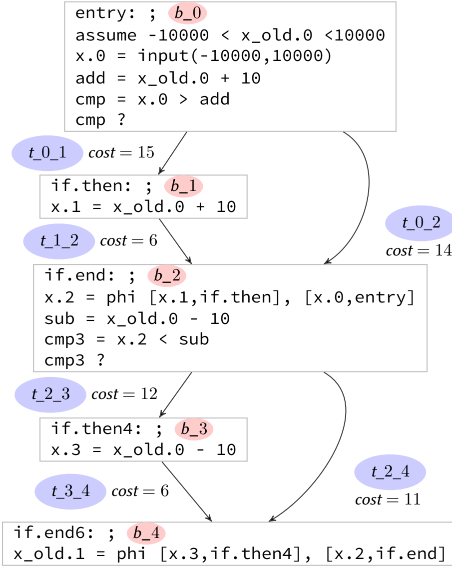
Step 2: encode control flow (Very similar to ILP)

$$
\begin{array}{ll}
\wedge & b\_0 = b\_4 = \mathit{true} \\
\wedge & b\_1 = t\_0\_1 \\
\wedge & b\_2 = (t\_0\_2 \vee t\_1\_2) \\
\wedge & \vdots \\
\wedge & \vdots \\
\wedge & t\_0\_1 = (b\_0 \wedge (x.0 > \mathit{add})) \\
\wedge & \vdots \\
\wedge & \vdots
\end{array}
$$



Step 3: encode timings

$$
\begin{aligned}
\wedge\quad & c\_0\_1 = (\mathit{if}(t\_0\_1) \text{ then } 15 \text{ else } 0) \\
\wedge\quad & c\_0\_2 = (\mathit{if}(t\_0\_2) \text{ then } 14 \text{ else } 0) \\
\wedge\quad & \vdots \\
\wedge\quad & \vdots \\
\wedge\quad & \vdots \\
\wedge\quad & \mathit{cost} = (c\_0\_1 + c\_0\_2 + c\_1\_2 \\
& \qquad + c\_2\_3 + c\_2\_4 + c\_3\_4)
\end{aligned}
$$



1 satisfying assignment
$\leftrightarrow$ 1 program trace:

$b\_0 = b\_1 = b\_2 = b\_4 = \text{true}$

$b\_3 = \text{false}$

$t\_0\_1 = t\_1\_2 = t\_2\_4 = \text{true}$

$t\_0\_2 = t\_2\_3 = t\_3\_4 = \text{false}$

$\text{x\_old.0} = 50$

$\text{x.0} = 61$

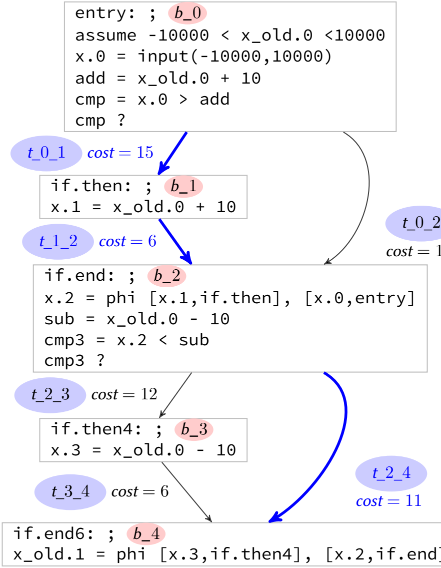
$\text{add} = 60$

$\text{x.1} = 60$

$\text{x.2} = 60$

$\text{sub} = 40$

$\color{red}{\textbf{cost} = \mathbf{32}}$



1 satisfying assignment
$\leftrightarrow$ 1 program trace:

$b\_0 = b\_1 = b\_2 = b\_4 = \text{true}$

$b\_3 = \text{false}$

$t\_0\_1 = t\_1\_2 = t\_2\_4 = \text{true}$

$t\_0\_2 = t\_2\_3 = t\_3\_4 = \text{false}$

$\text{x\_old.0} = 50$

$\text{x.0} = 61$

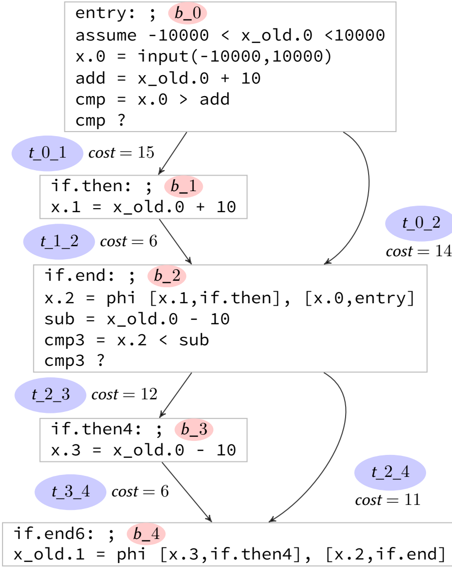
$\text{add} = 60$

$\text{x.1} = 60$

$\text{x.2} = 60$

$\text{sub} = 40$

cost = 32



We want the trace with the highest cost

Mixed approach to optimization

(see in MathSAT)

Binary search Start with lower and upper bound,
divide the interval in two
test for satisfiability above the midpoint.

If seeking integer value, termination ensues.

Local search Find a polyhedron $\bigwedge l_i \implies \phi$, optimize locally in $\bigwedge l_i$, get a new bound.

Diamonds

Corresponds to sequence of n "if-then-else":

```
if (b[i]) { timing 2 } else { timing 3 }
if (b[i]) { timing 3 } else { timing 2 }
```

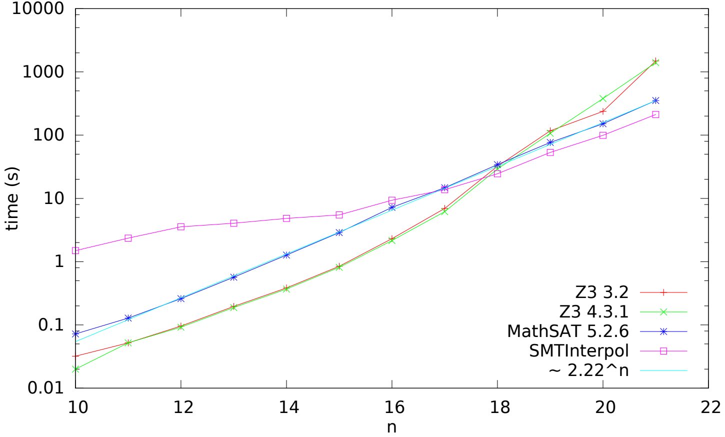
$D(n)$ the unsatisfiable formula:

$$\text{for } 0 \le i < n \begin{cases} x_i - t_i \le 2 \\ y_i - t_i \le 3 \\ (t_{i+1} - x_i \le 3) \vee (t_{i+1} - y_i \le 2) \end{cases}$$

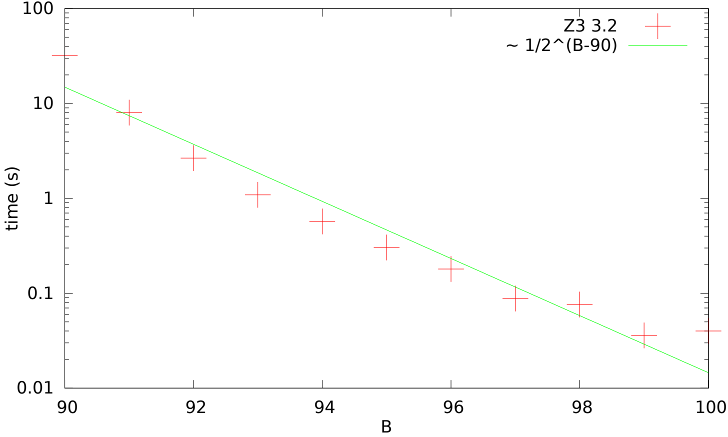
$$t_n - t_0 > 5n$$

Schedule

Behavior of SMT-solvers



Cost increases near bound



DPLL(T) on diamonds

Will enumerate each combination of disjuncts =
All terms in disjunctive normal form

Fundamental limitation: can only use **atoms from original formula**.

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

- WCET example
- CDCL explosion
- Abstract CDCL (ACDCL)
- Model-construction satisfiability calculus (MCSAT)

Abstract CDCL

DPLL / CDCL assign truth values to Booleans

$\downarrow$ *generalization*

ACDCL assigns truth values to Booleans and intervals to reals (or elements from an abstract domain)

e.g. if current assignment $x \in [1, +\infty)$ and $y = [4, 10]$
constraint $z = x - y \rightsquigarrow x \in [-9, +\infty)$

If too coarse, **split** intervals.
Akin to **constraint programming**.

Learning in ACDCL

Constraints $(y = x) \wedge (z = x \cdot y) \wedge (z \leq -1)$
Search context $x \leq -4$, **contradiction**.

Contradiction ensured by $x < 0$ **weaker** than search context.

Learn $x < 0$. Predicate **not in original formula**.

(CDCL-style learning would only learn $x > -4$.)

Non unicity of learning

Choices $x \geq 10$ and $y \geq 10$ constraint $x + y < 10$. Possible

generalizations:

- $x \geq 0$ and $y \geq 10$
- $x \geq 5$ and $y \geq 5$
- $x \geq 10$ and $y \geq 0$

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

- WCET example
- CDCL explosion
- Abstract CDCL (ACDCL)
- Model-construction satisfiability calculus (MCSAT)

MCSAT

In DPLL(T), assign only to Booleans and atoms from original formula.
In MCSAT, assign to propositional atoms *and* numeric variables $x_1, \ldots, x_n, \ldots$

When finding an impossibility when trying to assign to x_{n+1}, derive a general impossibility on $x_1, \ldots, x_n$ (**partial projection**).

Example: diamonds

$$\text{for } 0 \leq i \leq 2 \begin{cases} x_i - t_i \leq 2 \\ y_i - t_i \leq 3 \\ t_{i+1} - x_i \leq 3 \vee t_{i+1} - y_i \leq 2 \end{cases}$$

$$t_0 = 0$$

$$t_3 \geq 16$$

Pick $t_0 \mapsto 0$, $t_1 - x_0 \leq 3 \mapsto \mathbf{t}$, $x_0 \mapsto 0$,
$t_1 \mapsto 0$, $t_2 - x_1 \leq 3 \mapsto \mathbf{t}$, $x_1 \mapsto 0$,
$t_2 \mapsto 0$, $t_3 - x_2 \leq 3 \mapsto \mathbf{t}$, $x_2 \mapsto 0$.

No way to assign to x_3!
Because $x_2 \mapsto 0$ and $t_3 - x_2 \leq 3$ and $t_3 \geq 16$.

Analyze the failure

$x_2 \mapsto 0$ fails due to a **more general reason** (Fourier-Motzkin)

$$\left\{\begin{array}{l} t_3 - x_2 \leq 3 \\ t_3 \geq 16 \end{array}\right. \implies x_2 \geq 13$$

Possible to learn

$$t_3 - x_2 > 3 \vee x_2 \geq 13$$

Retract $x_2 \mapsto 0$.

Backtracking

We have learnt $t_3 - x_2 > 3 \vee x_2 \geq 13$.
$t_3 - x_2 \leq 3$ still assigned.

$$\left\{ x_2 \geq 13 x_2 - t_2 \leq 2 \quad \implies \quad t_2 \geq 11\right.$$

Thus learn

$$t_3 - x_2 > 3 \vee t_2 \geq 11$$

$t_3 - x_2 \leq 3 \mapsto \mathbf{t}$ retracted.

Continuation

Same reasoning for $t_3 - x_2 \leq 3 \mapsto \mathbf{f}$ yields by learning

$$t_3 - x_2 \leq 3 \vee t_2 \geq 11$$

Thus

$$\begin{cases} t_3 - x_2 > 3 \vee t_2 \geq 11 \\ t_3 - x_2 \leq 3 \vee t_2 \geq 11 \end{cases} \implies t_2 \geq 11$$

One learns $t_2 \geq 11$.
Then $t_1 \geq 6$ similarly.

But then no satisfying assignment to t_0!

NLSAT

(Dejan Jovanović, Leonardo De Moura)
MCSAT for **non-linear arithmetic**

Partial projection: Fourier-Motzkin replaced by partial **cylindrical algebraic decomposition**.

Nonexhaustive list of SMT-solvers

See also `http://smtlib.cs.uiowa.edu/`
`http://smtlib.cs.uiowa.edu/solvers.shtml`

Free

- Z3 (Microsoft Research)
  `https://github.com/Z3Prover`
- Yices (SRI International)
  `http://yices.csl.sri.com/`
- CVC4 `http://cvc4.cs.nyu.edu/web/`

Non-free

- MathSAT (Fundazione Bruno Kessler)
  `http://mathsat.fbk.eu/`

Schedule

Quantifier elimination

Over $\mathbb{Z}$ or $\mathbb{Q}$ or $\mathbb{R}$,

$$\forall y\, y \leq x \implies y \leq 1$$

is equivalent to

$$x \leq 1$$

Finding an equivalent formula without quantifiers = quantifier elimination

Note: quantifier elimination algorithm + decidable ground formulas
$\implies$ decidability

Schedule

Resolution

A formula in CNF $\bigwedge_i C_i$ = a set $\{C_1, \ldots, C_n\}$ of **clauses**.

Assume:

- no redundant literals in a clause (e.g. $a \vee a \vee b$)
- no trivially true clauses (e.g. $a \vee \neg a \vee b$).

For clauses where a appears, apply **resolution**:

$$\frac{C'_i \vee a \quad C'_j \vee \neg a}{C'_j \vee C'_j}$$

Th: the result (clauses without a) is equivalent to the projection on variables except for a

$$C'_1 \wedge \cdots \wedge C'_{n'} \equiv \exists a \left(C_1 \wedge \cdots \wedge C_n\right)$$

Some remarks on resolution

- Not efficient if applied blindly.
- May be used as **simplification** if not inflating the set too much.
- $\leq 3^{|V|}$ different clauses, thus **termination**.
- Detect subsumption: do not store $a \vee b \vee c$ in addition to $a \vee b$.
- Eliminate all variables: obtain the empty clause (**f**) iff $\bigwedge_i C_i$ unsatisfiable.
- (More on this later) DPLL/CDCL SAT-solvers finding "unsat" can give a resolution proof (more clever than blind search)

Other method

$$\exists x\, F(x) \equiv F(0) \vee F(1)$$
$$\forall x\, F(x) \equiv F(0) \wedge F(1)$$

Generalizes to any finite structure.

Again, explosive complexity !

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

- Booleans
- Projecting conjunctions
- Substitution methods

Remarks

- $\forall x\, F \equiv \neg \exists x \neg\, F$
- $\exists x\, (F_1 \vee F_2) \equiv (\exists x\, F_1) \vee (\exists x\, F_2)$
- $\exists x\, F \equiv \exists x\, F'$ where F' DNF of F

All cases boil down to $\exists x_1 \ldots x_n\, C$ where C conjunction.

Geometrically

C conjunction of linear inequalities.

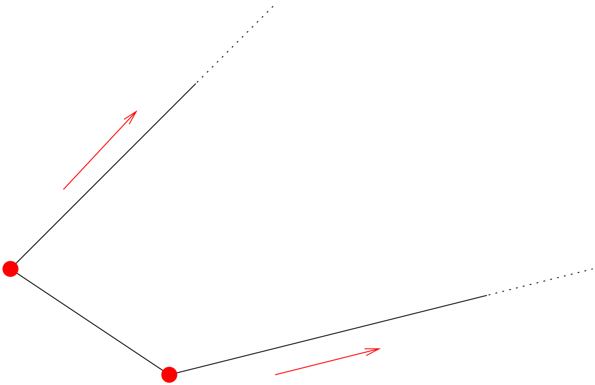
Strict inequalities: $L(x, y, \dots) < B$ into $L(x, y, \dots) + \epsilon \geq B$.

Closed convex polyhedron in $x, y, \dots, \epsilon$.

Either represented by **constraints** (= faces) or **generators** (= vertices, + rays and lines for unbounded).

One projection method: go to generators, project them, move back to constraints.

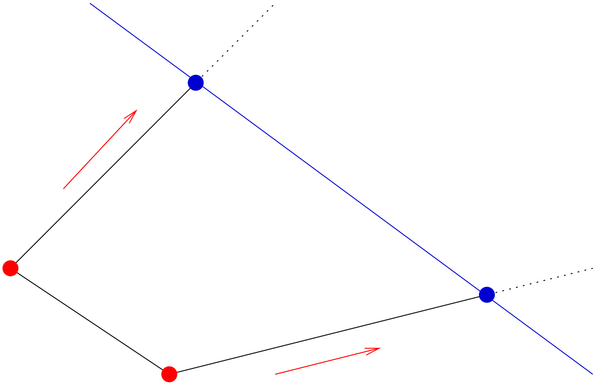
Constraints vs generators



3 constraints $\equiv$ 2 vertices + 2 rays

Chernikova's algorithm

Often applied following Le Verge's remarks.
`http://www.irisa.fr/polylib/document/cher.ps.gz`
Compute generators from constraints : conjoin with constraints one by one.



Chernikova's algorithm, reversed

Duality constraints vs generators
Dual polyhedron: reverse constraints and generators

Fourier-Motzkin

Takes a list L of linear inequalities and a variable x. Split L into:

- L_+, where x has positive coefficient ($n.x + \cdots \leq b$), thus $\equiv x \leq \frac{1}{n}.(b - \ldots)$
- L_-, where x has negative coefficient ($(-n).x + \cdots \leq b$), thus $\equiv x \geq \frac{1}{n}.(b - \ldots)$
- L_0, without x

Otherwise said

$$\max_i l_i^-(y, \ldots) \leq x \leq \min_j l_j^+(y, \ldots)$$

Elimination

$$\exists x \max_i l_i^-(y,\dots) \leq x \leq \min_j l_j^+(y,\dots)$$

iff $\max_i l_i^-(y,\dots) \leq \min_j l_j^+(y,\dots)$

iff for all i and j $l_i^-(y,\dots) \leq l_j^+(y,\dots)$

Elimination:

- Copy L_0
- For all pair $(l_i^-(y,\dots) \geq x, x \leq l_j^+(y,\dots)) \in L_- \times L_+$ produce $l_i^-(y,\dots) \leq l_j^+(y,\dots)$.

Example

Eliminate y from:
$x + y \leq 1 \wedge -x + y \leq 1 \wedge x - y \leq 1 \wedge -x - y \leq 1$

$$x + y \leq 1 \wedge x - y \leq 1 \rightsquigarrow x \leq 2 \equiv x \leq 1$$
$$x + y \leq 1 \wedge -x - y \leq 1 \rightsquigarrow 0 \leq 2$$
$$-x + y \leq 1 \wedge x - y \leq 1 \rightsquigarrow 0 \leq 2$$
$$-x + y \leq 1 \wedge -x - y \leq 1 \rightsquigarrow -2x \leq 2 \equiv x \geq -1$$

Note: generates **trivial** constraints and more generally **redundant** constraints.

Constraint growth

Project 1 variable: if $n/2$ positive and $n/2$ negative constraints, then $n^2/4$ constraints in the output.

(Heuristic: start with the dimensions minimizing # positive constraints $\times$ # negative constraints)

For p projected dimensions: bound in n^{2^p}.

But McMullen's bound (# dimension-k faces of a polyhedron with v vertices in d-dimension space) yields a single exponential bound!

Anything above is **redundant constraints**.

Elimination of redundant constraints

- Syntactic criteria (cf Simon & King, SAS 2005), e.g. $a_1 x_1 + \cdots + a_n x_n \leq B$ eliminated by $a_1 x_1 + \cdots + a_n x_n \leq B'$ with $B' \leq B$
- **Linear programming**: if we have C and add C' $(a_1 x_1 + \cdots + a_n x_n \leq B)$,
  - test emptiness of $C \wedge \neg C'$ by linear programming
  - or maximize $a_1 x_1 + \cdots + a_n x_n$ w.r.t C and keep C' if B less than the optimum
- Or ray-tracing (Maréchal & Périn, 2017) `https://hal.archives-ouvertes.fr/hal-01385653/document`

Improvements on projection-based methods

Convert F to DNF then project?

Rather (Monniaux, LPAR 2008)

- Extract a conjunction $C \Rightarrow F$ of atoms of F (see SMT-solving).
- Extract maximal conjunction C' s.t. $C \Rightarrow C' \Rightarrow F$. From SMT-solving and/or unsat-core.
- Project C' into $\pi(C')$, add to output F'.
- Conjoin $\neg\pi(C')$ to F.

and improvements around that theme.

Schedule

Basic ideas of substitution methods

Obvious: if x is in a finite domain $\{k_1, \ldots, k_n\}$ then $\exists x\, F \equiv F[x \mapsto k_1] \vee \cdots \vee F[x \mapsto k_n]$.

Nontrivial extension: For certain logics, can use k_i functions of free variables of the formula.

Known as substitution (or virtual substitution) methods.

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

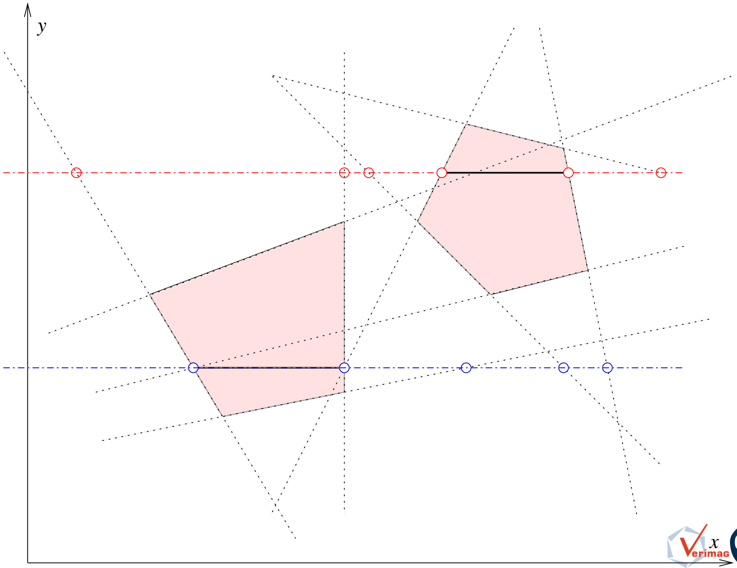
Quantifier elimination
- Booleans
- Projecting conjunctions
- Substitution methods

Linear inequalities over the reals

Impossible to distinguish $\mathbb{Q}$, $\mathbb{R}$, etc.

Axioms: totally ordered group + scheme « $\forall x \exists y\ x = \mathbf{n}.x$ » for any natural *n*.

Test points



Ferrante & Rackoff's method

$\exists x\, F$ is true iff

- true for $x \to -\infty$, $x \to +\infty$,
- true for all x intersection point or median to intersections

Let $x_1, \dots, x_n$ the intersections as functions of $y, z, \dots$

$$\exists x\, F \equiv F[x \mapsto -\infty] \vee F[x \mapsto +\infty] \vee \bigvee_i F[x \mapsto x_i] \vee \bigvee_{i<j} F\left[x \mapsto \frac{x_i + x_j}{2}\right]$$

Quadratic number of substitutions.

Loos & Weisfpenning's method

F in negation normal form, leaves are $x \geq \ldots$, $x \leq \ldots$, $x > \ldots$, $x < \ldots$

$\exists x\, F$ true iff

- true when $x \rightarrow -\infty$
- true for all $x = x_i$ given $x \geq x_i(y, \ldots)$
- true for all $x = x_i + \epsilon$ given by $x > x_i(y\, dots)$

ϵ infinitesimal, $x \geq t + \epsilon$ means $x > t$

$$\exists x\, F \equiv F[x \mapsto -\infty] \vee F[x \mapsto +\infty] \vee \bigvee_i F[x \mapsto x_i] \vee \bigvee_j F[x \mapsto x'_i + \epsilon]$$

Linear number of substitutions.

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination
- Booleans
- Projecting conjunctions
- Substitution methods

Presburger arithmetic

$+$, $-$, $\geq$ (+multiplication by constant)

Does it admit quantifier elimination?

Presburger arithmetic

$+, -, \geq$ (+multiplication by constant)

Does it admit quantifier elimination?

No: $\exists x\; y = 2x \not\equiv$ a quantifier-free formula

If adding an infinity of predicates $n|x$ (n natural constant) then admits quantifier elimination.

Cooper's method

Same idea as Loos & Weisfpenning

Put F in NNF: atoms are $h.x \leq \ldots$, $h.x \geq \ldots$, $h.x < \ldots$, $h.x > \ldots$, $h.x = \ldots$, $\neq$, $n \mid \ldots$, $n \nmid \ldots$
Rewrite $=$, $\neq$, $\geq$, $\leq$ into $>$, $<$.
Any atom is thus $h.x < t$, $h.x > t$, $n \mid h.x + t$ or $n \nmid h.x + t$ (t without x)

Let m be the least common multiple of h. Scale atoms such that $h.x$ into $m.x$.
Replace $m.x$ by x' and conjoin $m \mid x'$.

Get F'. Any atom is $x' < t$, $x' > t$, $n \mid x' + t$ or $n \nmid x' + t$ (t without x')

Some remark

Let δ the least common multiple of n in $n \mid x' + t$ or $n \nmid x' + t$.
Divisibility predicates have δ-periodic truth value.

Fix $y, z, \ldots$. There is a solution x' iff

- there is an infinity of solutions $\rightarrow -\infty$
- or there is a least solution

Solutions to $-\infty$

(Fix $y, z, \ldots$)
Case when for all M there is a solution $x' < M$.
There are thus solutions x'

- making true all atoms $x' < t$
- making false all atoms $x' > t$.

Replace $x' < t$ (etc.) by $\mathbf{t}$, $x' > t$ by $\mathbf{f}$.
Obtain $F_{-\infty}$. Only divisibility atoms.

The problem is now δ-periodic, thus take

$$F'[x' \mapsto 1] \vee \cdots \vee F'[x' \mapsto \delta]$$

Other solutions

There is a least solution x'.
Can only exist if $x' > t$ has become true.

Any x' is solution if it satisfies the same inequalities and the same divisibility predicates.
By δ-periodicity:

Test $F'[x' \mapsto t + 1] \ldots F'[x' \mapsto t + \delta]$.

Finally:

$$\exists x\, F \equiv \exists x'\; F' \equiv \quad \bigvee_{j=1}^{\delta} F'_{-\infty}[x' \mapsto j] \vee \bigvee_{j=1}^{\delta} \bigvee_{t \in B} F'[x' \mapsto b + j]$$

Some optimizations

Replace x by $-x$ and apply the same process?

If several variables are to be eliminated: move $\exists y_1 \dots y_n$ into the disjunction terms $\bigvee_{j=1}^{\delta} \bigvee_{t \in B} F'[x' \mapsto b + j]$.

Eliminate in $\exists y_1 \dots y_n F'[x' + j]$ where j extra variable, then replace j.

In disjunctions, test using SMT-solving if the formula is satisfiable.

(see e.g. works by Nikolaj Bjørner)

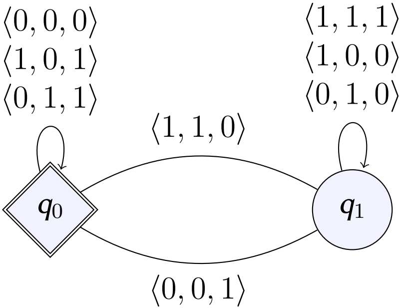
Schedule

- Introduction
- Propositional logic
- First-order logic
- Applications
- Beyond DPLL(T)
- Quantifier elimination
  - Booleans
  - Projecting conjunctions
  - Substitution methods

Encoding into regular languages

Formula with n free variables in $\mathbb{N}$
$\rightsquigarrow$ finite automaton recognizing n-tuples of binary words
$\rightsquigarrow$ finite automaton recognizing words over $\{0,1\}^n$

Example : $z = x + y$, recognize (x, y, z)



Constructions

- Constants: easy.
- $\mathbb{Z}$: use a sign bit, or encode ≥ 0 into even numbers and < 0 into odd numbers.
- Successor, addition etc.: propagate **carry** inside the automaton state.
- $\wedge$: intersection of regular languages
- $\vee$: union of regular languages
- $\neg$: complement
- $\exists v_i\ F$: make transitions depending on the i-th input nondeterministic

Check for satisfiability: does the automaton accept words?

Richness of the logic

We have encoded Presburger arithmetic into finite automata.
Are there finite automata encoding non-Presburger formulas?

Richness of the logic

We have encoded Presburger arithmetic into finite automata.
Are there finite automata encoding non-Presburger formulas?

Yes: 0^*1 encodes $\{2^n \mid n \in \mathbb{N}\}$

Difficult to take an automaton and convert it into Presburger.

Costly procedures. Implementations: MONA, LIRA…

Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

- Booleans
- Projecting conjunctions
- Substitution methods

Theory of the real closed fields

Totally ordered field such that

- any positive number has a square root
- axiom scheme indexed by $P \in \mathbb{Z}[X]$ of odd degree, then P has at least one root.

Cannot distinguish $\mathbb{R}$ from algebraic reals.

NB: Quantifier elimination with all steps provable inside the theory
$\Rightarrow$ any closed formula is decidable
$\Rightarrow$ all models of the theory satisfy the same formulas

Theorem

Tarski (1951) The theory of real closed fields admits quantifier elimination.

From the proof one can extract an algorithm with huge complexity.

Now: algorithm of **cylindrical algebraic decomposition** (Collins) + many improvements

Difficult to implement, few implementations (QEPCAD, Mathematica, partial in Microsoft Z3, partial in Yices?)

Muchnik's proof

Start: polynomials $P_1, \ldots, P_m \in \mathbb{Z}[X, Y_1, \ldots, Y_n]$ with imposed signs $\sigma_1, \ldots, \sigma_m \in \{-, 0, +\}$.

End: polynomials $P'_1, \ldots, P'_{m'} \in \mathbb{Z}[Y_1, \ldots, Y_n]$ such that imposing their sign yields a unique **sign diagram** for $P_1, \ldots, P_m$ w.r.t X.

Try all combinations of signs for $P'_1, \ldots, P'_{m'}$, keep those for which at least one suitable zone appears for X.

Saturation

Saturate the set of input polynomials by:

- derivation: given P, add $\mathrm{d}P/\mathrm{d}X$
- extraction of leading coefficient: from $\sum_{k=0}^{d} a_k X^k$ ($a_k \in \mathbb{Z}[Y_1, \dots, Y_m] \setminus \{0\}$) get a_k
- removal of leading coefficient: from $\sum_{k=0}^{d} a_k X^k$ get $\sum_{k=0}^{d-1} a_k X^k$

and "modified remainder" (modified to avoid non-integer coefficients) if $A, B \in \mathbb{Z}[X, Y_1, \dots, Y_m]$, $\deg A \geq \deg B$, and D leading coefficient of B, there exist unique Q and R s.t. $D^{\deg A - \deg B + 1}.A = QB + R$; return R.

Group polynomials into "strata" by application of the last rule.

Sign diagram

Each $\gamma_1 < \gamma_2 < \ldots$ corresponds to a root of at least one of the polynomials $P_1, \ldots, P_m$.

For each P_i and interval $]-\infty, \gamma_1[$, $\{\gamma_1\}$, $]\gamma_1, \gamma_2[$, $\{\gamma_2\}$, ...give a sign $(-, 0, +)$.

	$-\infty$	γ_1		γ_2		γ_3	$+\infty$
b	$+$	$+$	$+$	$+$	$+$	$+$	$+$
c	$+$	$+$	$+$	$+$	$+$	$+$	$+$
$4c - b^2$	$-$	$+$	$-$	$-$	$-$	$-$	$-$
$2x + b$	$-$	$-$	$-$	0	$+$	$+$	$+$
$x^2 + bx + c$	$+$	0	$-$	$-$	$-$	0	$+$

Idea of the proof

First impose the signs of the polynomials of degree 0 in X.

b	$+$
c	$+$
$4c - b^2$	$-$

The other polynomials

The sign of its derivative imposes the behavior $2x + b$, thus a root γ_2

	$-\infty$	γ_2	$+\infty$
b	$+$	$+$	$+$
c	$+$	$+$	$+$
$4c - b^2$	$-$	$-$	$-$
$2x + b$	$-$	0	$+$

From one stratum to one higher stratum

$x^2 + bx + c$ is $+$ when $x \to \pm\infty$

From $4(x^2 + bx + c) = (2x + b)(2x + b) + (4c - b^2)$ get the sign of $x^2 + bx + c$ at $x = \gamma_2$

	$-\infty$	γ_2	$+\infty$
b	$+$	$+$	$+$
c	$+$	$+$	$+$
$4c - b^2$	$-$	$-$	$-$
$2x + b$	$-$	0	$+$
$x^2 + bx + c$	$+$	$-$	$+$

Adding other roots

By continuity, need extra roots γ_1 and γ_3

	$-\infty$	γ_1		γ_2		γ_3	$+\infty$
b	$+$	$+$	$+$	$+$	$+$	$+$	$+$
c	$+$	$+$	$+$	$+$	$+$	$+$	$+$
$4c - b^2$	$-$	$+$	$-$	$-$	$-$	$-$	$-$
$2x + b$	$-$	$-$	$-$	0	$+$	$+$	$+$
$x^2 + bx + c$	$+$	0		$-$		0	$+$

Complete

There can be no more roots of $x^2 + bx + c$ in $]\gamma_1, \gamma_2[$ or $]\gamma_2, \gamma_3[$ otherwise the derivative should have a zero.

	$-\infty$	γ_1		γ_2		γ_3	$+\infty$
b	$+$	$+$	$+$	$+$	$+$	$+$	$+$
c	$+$	$+$	$+$	$+$	$+$	$+$	$+$
$4c - b^2$	$-$	$+$	$-$	$-$	$-$	$-$	$-$
$2x + b$	$-$	$-$	$-$	0	$+$	$+$	$+$
$x^2 + bx + c$	$+$	0	$-$	$-$	$-$	0	$+$

More details?

Michaux & Ozturk, *Quantifier elimination following Muchnik*

This algorithm cannot be used except on very small systems but has a simple proof.

More involved mathematics for **cylindrical algebraic decomposition** but same general idea of “projecting” behaviors.

Schedule

Conclusion

Quantifier elimination

- Rather easy on linear theory of reals.
- Harder on linear theory of integers (Presburger) — see Fischer & Rabin, 1974 for lower bound on costs.
- Painful in another way on polynomial real arithmetic (real closed fields).
- Impossible in general on polynomial integer arithmetic (undecidability) — we'll see it.

Schedule

Introduction

Propositional logic

First-order logic

Applications

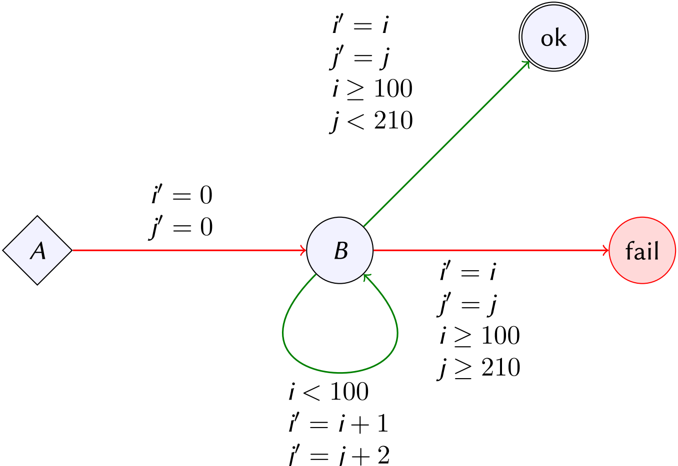
Beyond DPLL(T)

Quantifier elimination

Interpolants

Predicate abstraction

```
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



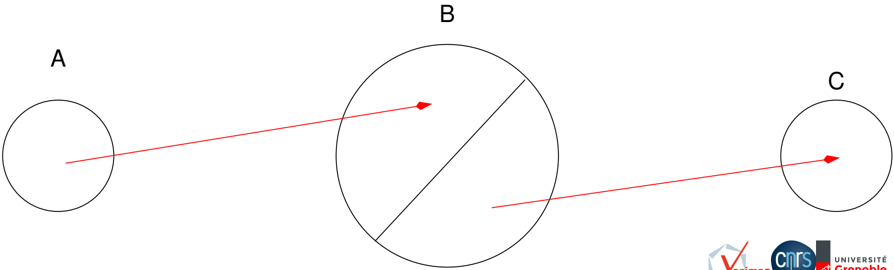
A bad counterexample

Try to find values for the red path:
$i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \wedge i \geq 100 \wedge j \geq 210$

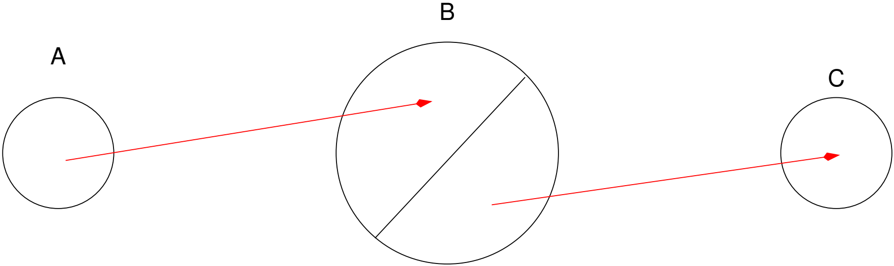
A bad counterexample

Try to find values for the red path:
$i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \wedge i \geq 100 \wedge j \geq 210$
UNSAT

Why wrong? Can move from one state in *A* to one state in *B*, from one state in *B* to one in "fail". But states in *B* not the same.



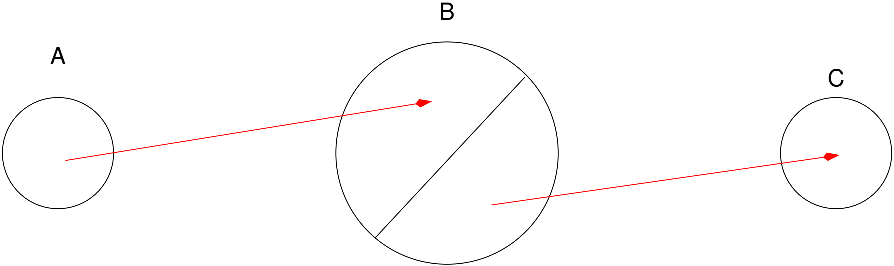
A refinement



Two step explanation for infeasible path:

- $i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \Rightarrow j_2 = 2i_2 \wedge i \geq 100$
- $j_2 = 2i_2 \wedge i_2 \geq 100 \Rightarrow j_2 < 210$

A refinement



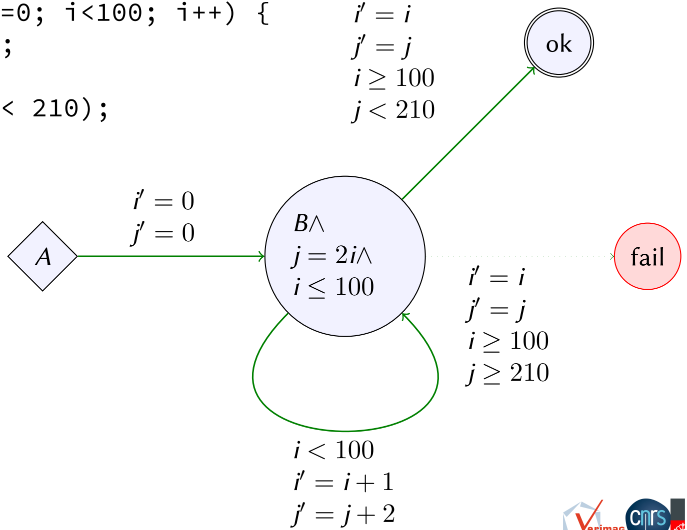
Two step explanation for infeasible path:

- $i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \Rightarrow j_2 = 2i_2 \wedge i \geq 100$
- $j_2 = 2i_2 \wedge i_2 \geq 100 \Rightarrow j_2 < 210$

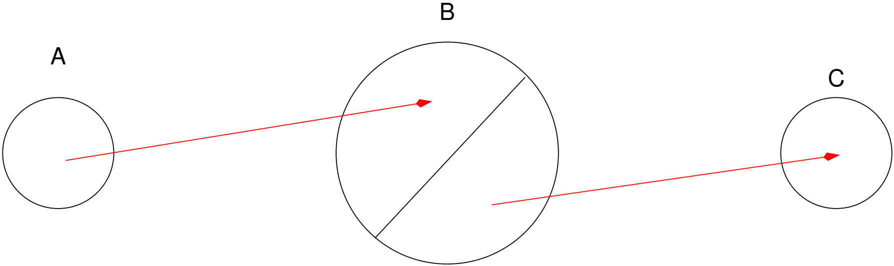
This is a **Craig interpolant**.

A good refinement

```
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



Another refinement



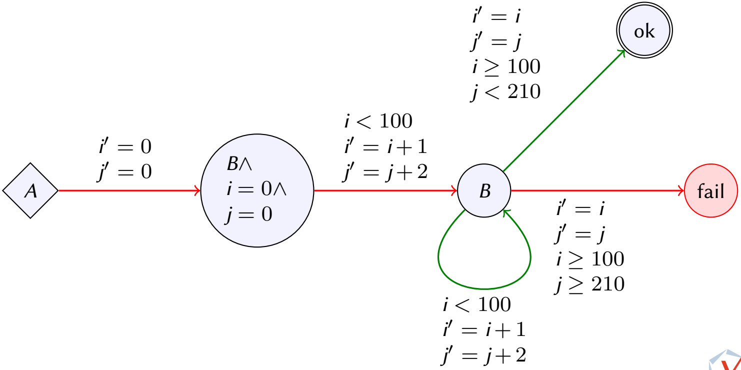
Two step explanation for infeasible path:

- $i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \Rightarrow i_2 = 0 \wedge j_2 = 0$
- $i_2 = 0 \wedge j_2 = 0 \Rightarrow j_2 < 210$

This is another **Craig interpolant**.

Another refinement

```
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



Further refinement

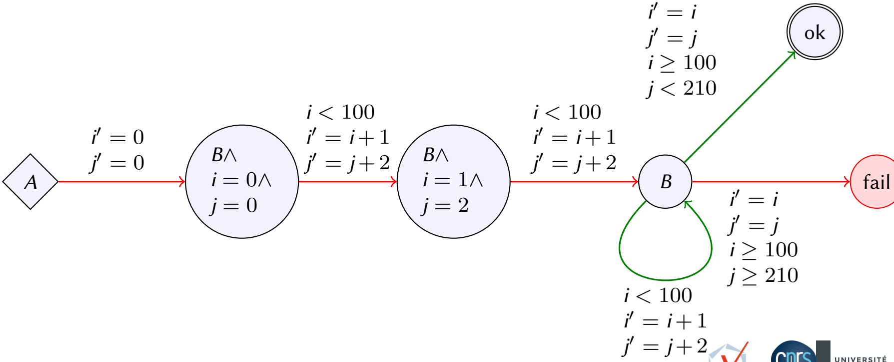
Two step explanation for infeasible path:

- $i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \Rightarrow i_2 = 0 \wedge j_2 = 0$
- $i_2 = 0 \wedge j_2 = 0 \wedge i_3 = i_2 + 1 \wedge j_3 = j_2 + 2 \Rightarrow i_3 = 1 \wedge j_3 = 2$
- $i_3 = 1 \wedge j_3 = 2 \Rightarrow j_2 < 210$

This is another **Craig interpolant**.

Further refinement

```
for(int i=0; i<100; i++) {
  j = j+2;
}
assert(j < 210);
```



Overfitting and convergence

- Interpolant $j = 2i \land i \leq 100$ (polyhedral inductive invariant) proves the property.
- Interpolants $i = 0 \land j = 0$, $i = 1 \land j = 2$, $i = 2 \land j = 4$ …(exact post-conditions) lead to non-termination.

Challenge: find "good" interpolants "likely" to become inductive
Problem similar to widening
McMillan: find "short" interpolants using few "magic" constants?

Problem statement

Suppose $A(x, y) \implies B(y, z)$
Obtain $I(y)$ such that $A \implies I \implies B$

I talks about **common variables**

If theory admits quantifier elimination, possibilities:

Stronger $\exists x\, A(x, y)$

Weaker $\forall z\, B(y, z)$

but they may be "too precise" (overfitting !)

Interpolants from proofs

Suppose $A \wedge B$ unsatisfiable (aka $A \implies \bar{B}$)
Obtain a resolution proof of $\mathbf{f}$, process proof to get interpolants (McMillan)

For a clause c, $g(c) = c$ keeping only **global symbols** (common to A and B). $[g(c)$ **partial interpolant** at c

Rules

(courtesy of Ken McMillan and Philipp Rümmer)

$$\frac{}{c \quad [c \downarrow B]}\ c \in A \qquad\qquad \frac{}{c \quad [\mathbf{t}]}\ c \in B$$

$$\frac{v \lor c \quad [I_1] \qquad \bar{v} \lor d \quad [I_2]}{c \lor d \quad [I_1 \lor I_2]}\ v \text{ does not occur in } B$$

$$\frac{v \lor c \quad [I_1] \qquad \bar{v} \lor d \quad [I_2]}{c \lor d \quad [I_1 \land I_2]}\ v \text{ occurs in } B$$

Correctness

$c \setminus B$ clause c without the literals occurring in B
$c \downarrow B$ clause c with only the literals occurring in B

In any such annotated proof at any node $c \quad [I_c]$

- $A \models I_c \vee (c \setminus B)$
- $B, I_c \models c \downarrow B$
- $I_c \preceq A$ and $I_C \preceq B$

In particular at the root!

With theories

Combining preceding rules with theory-specific rules

Can become complicated if the theory introduces mixed literals when solving
(= literals with variables from both A and B)
see Jürgen Christ's thesis

Linear real arithmetic: simple case

Interpolants in linear real arithmetic

$\neg\phi$ is a conjunction of inequalities $C_1 \wedge \cdots \wedge C_n$
C_i over $\vec{x}, \vec{y}, \vec{z}$ vectors
Collect C_i into A_i over $\vec{x}, \vec{y}$ and B_i over $\vec{y}, \vec{z}$

$\bigwedge_i A_i$ is a polyhedron over $\vec{x}, \vec{y}$
$A' = \exists \vec{x} \ \bigwedge_i A_i$ is a polyhedron over $\vec{y}$
$\bigwedge_i B_i$ is a polyhedron over $\vec{y}, \vec{z}$
$B' = \exists \vec{z} \ \bigwedge_i B_i$ is a polyhedron over $\vec{y}$

$A' \cap B' = \emptyset$

Find a separating hyperplane: $A' \models I_\phi$, $B' \models \neg I_\phi$

Difficulties

Solving certain theories involves adding new predicates
e.g. branching and cutting planes in linear integer arithmetic
some of these predicates may involve local variables from A and B
they should not be made global

Interpolation then more complicated
(see e.g. Jürgen Christ's thesis)

Criticism

The proof tree depends on heuristics and random choices (variables, polarities, restarts...).
The interpolant thus depends on them.
Interpolants get fed into a refinement loop

$\Longrightarrow$ **brittleness**

Search for "simpler", more "beautiful" interpolants?

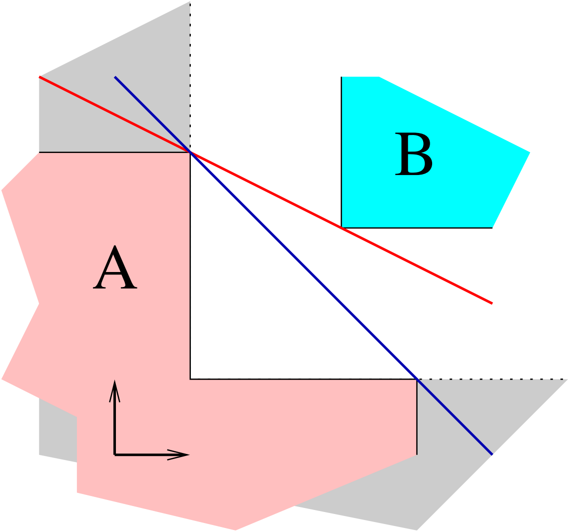
An interpolation problem

Interpolation problem $A \Rightarrow I$, $I \Rightarrow \neg B$:

$$\begin{aligned} A_1 &= x \leq 1 \wedge y \leq 4 & \qquad A_2 &= x \leq 4 \wedge y \leq 1 \\ A &= A_1 \vee A_2 & B &= x \geq 3 \wedge y \geq 3\,. \end{aligned} \tag{6}$$

SMTInterpol and MathSAT produce $I = x \leq 1 \vee y \leq 1$.

Simpler, more beautiful ones



How about $x + 2y \leq 9$, or $x + y \leq 5$?

Extensions

Trace interpolants

Tree interpolants (for Horn clauses)

Questions ?

For internships, theses etc.:
`http://www-verimag.imag.fr/~monniaux/`
`David.Monniaux@univ-grenoble-alpes.fr`