Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants
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

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants
Schedule

Introduction

Propositional logic
- Generalities
- Binary decision diagrams
- Satisfiability checking
- Applications

First-order logic

Applications

Beyond DPLL(T)
Quantifier-free propositional logic

\( \land \) (and), \( \lor \) (or), \( \neg \) (not) (also noted \( \overline{x} \))
Possibly add \( \oplus \) (exclusive-or)

\( t = \text{“true”}, f = \text{“false”} \)

Formula with variables: \((a \lor b) \land (c \lor \overline{a})\)

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

\[(a \lor b) \land c \rightarrow (a \land c) \lor (b \land c)\]

Inconvenience?
Disjunctive normal form

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

Obtained by distributivity

\[(a \lor b) \land c \rightarrow (a \land c) \lor (b \land c)\]

Inconvenience?
Exponential size in the input.
Conjunctive normal form

Disjunction of literals = clause
CNF = conjunction of clauses

Obtained by distributivity

\[(a \land b) \lor c \rightarrow (a \lor c) \land (b \lor c)\]
Negation normal form

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

\[
\neg(a \lor b) \rightarrow (\neg a) \land (\neg b)
\]

\[
\neg(a \land b) \rightarrow (\neg a) \lor (\neg b)
\]

Makes formula “monotone” with respect to \( f < t \) ordering.
Applications

Verification of combinatorial circuits

Equivalence of two circuits
Schedule

Introduction

Propositional logic
   Generalities
   Binary decision diagrams
   Satisfiability checking
   Applications

First-order logic

Applications

Beyond DPLL(T)
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 \((a \land c) \lor (b \land c)\):

```
\[
\begin{array}{cccc}
  & a & 1 & \\
  & 0 & \quad & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
  b & & & \\
  & 0 & \quad & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
  c & 0 & 0 & 1 \\
\end{array}
\]
```
Removing useless nodes

Silly to keep two identical subtrees:

\[ a \]
\[ \begin{array}{c}
0 \\
1
\end{array} \]
\[ b \]
\[ \begin{array}{c}
0 \\
1
\end{array} \]
\[ c \]
\[ \begin{array}{c}
0 \\
0
\end{array} \]
\[ c \]
\[ \begin{array}{c}
0 \\
1
\end{array} \]
\[ c \]
\[ \begin{array}{c}
0 \\
1
\end{array} \]
\[ c \]
\[ \begin{array}{c}
0 \\
1
\end{array} \]

identical

identical
Compression

identical
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 $\land, \lor$, etc.
Binary BDD operations

Operations $\wedge$, $\lor$: 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 $O(|a| \cdot |b|)$ where $|x|$ the size of DAG $x$
Fixed point solving

\[ B_0 = \text{initial states} \]
\[ B_1 = B_0 + \text{reachable in 1 step from } B_0 \]
\[ B_2 = B_1 + \text{reachable in 1 step from } B_1 \]
\[ B_3 = B_2 + \text{reachable in 1 step from } B_2 \]
\[ \vdots \]

converges in finite time to \( R = \text{reachable states} \)
Iteration sequence

\[ B_{k+1}(x'_1, \ldots, x'_n) = \exists x_1 \ldots x_n \, B_k(x_1, \ldots, x_n) \land \tau(x_1, \ldots, x_n, x'_1, \ldots, x'_n) \]

Needs

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

e.g. NuSMV, NuXMV in BDD mode
Schedule

Introduction

Propositional logic
- Generalities
- Binary decision diagrams
- Satisfiability checking
- Applications

First-order logic

Applications

Beyond DPLL(T)
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


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

\[(a \lor b) \land (c \lor d) \longrightarrow (a \land c) \lor (a \land d) \lor (b \land c) \lor (b \land d)\]
Tseitin encoding

Add extra variables

\[((a \land \bar{b} \land \bar{c}) \lor (b \land c \land \bar{d})) \land (\bar{b} \lor \bar{c})\].

Assign propositional variables to sub-formulas:

\[
e \equiv a \land \bar{b} \land \bar{c} \quad f \equiv b \land c \land \bar{d} \quad g \equiv e \lor f
\]
\[
h \equiv \bar{b} \lor \bar{c} \quad \phi \equiv g \land h;
\]
Tseitin encoding

\[ e \equiv a \land \overline{b} \land \overline{c} \quad f \equiv b \land c \land \overline{d} \quad g \equiv e \lor f \]
\[ h \equiv \overline{b} \lor \overline{c} \quad \phi \equiv g \land h; \]

turned into clauses

\begin{align*}
\overline{e} \lor a & \quad \overline{e} \lor \overline{b} & \quad \overline{e} \lor \overline{c} & \quad \overline{a} \lor \overline{b} \lor \overline{c} \lor e \\
\overline{f} \lor b & \quad \overline{f} \lor c & \quad \overline{f} \lor 
\overline{d} & \quad \overline{b} \lor \overline{c} \lor \overline{d} \lor f \\
\overline{e} \lor g & \quad \overline{f} \lor g & \quad \overline{g} \lor e \lor f \\
b \lor h & \quad c \lor h & \quad h \lor \overline{b} \lor \overline{c} \\
\overline{\phi} \lor g & \quad \overline{\phi} \lor h & \quad \overline{g} \lor \overline{h} \lor \phi & \quad \phi
\end{align*}
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 \( \neg b \), clause \( \neg a \lor b \lor c \) yields \( c \)

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

\[
\begin{align*}
\bar{e} \lor a & \quad \bar{e} \lor \bar{b} & \quad \bar{e} \lor \bar{c} & \quad \bar{a} \lor \bar{b} \lor \bar{c} \lor \bar{e} \\
\bar{f} \lor \bar{b} & \quad \bar{f} \lor \bar{c} & \quad \bar{f} \lor \bar{d} & \quad \bar{b} \lor \bar{c} \lor \bar{d} \lor \bar{f} \\
\bar{e} \lor \bar{g} & \quad \bar{f} \lor \bar{g} & \quad \bar{g} \lor \bar{e} \lor \bar{f} & \quad b \lor h \\
\bar{c} \lor \bar{h} & \quad \bar{h} \lor \bar{b} \lor \bar{c} & \quad \bar{g} \lor \bar{h} \lor \phi & \quad \phi \\
\phi \lor \bar{g} & \quad \phi \lor \bar{h} & \quad \bar{g} \lor \bar{h} \lor \phi & \quad \phi
\end{align*}
\]

From unit clause \( \phi \)

\[
\begin{align*}
\phi \lor \bar{g} \to g & \quad \phi \lor \bar{h} \to h & \quad \bar{g} \lor \bar{h} \lor \phi \text{ removed}
\end{align*}
\]

Now \( g \) and \( h \) are \( t \),

\[
\begin{align*}
\bar{e} \lor \bar{g} \text{ removed} & \quad \bar{f} \lor \bar{g} \text{ removed} & \quad b \lor \bar{h} \text{ removed} \\
c \lor \bar{h} \text{ removed} & \quad \bar{g} \lor \bar{e} \lor \bar{f} \to \bar{e} \lor \bar{f} & \quad \bar{h} \lor \bar{b} \lor \bar{c} \to \bar{b} \lor \bar{c}
\end{align*}
\]
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. \( a \land \overline{b} \land \overline{c} \land d \land H \Rightarrow f \)

**Learn** clause = negation of bad hypotheses, implied by \( H \):

\[
\overline{a} \lor b \lor \overline{c} \lor \overline{d}
\]

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 $C_1 \lor \bar{a}$ from $H_1, \ldots, H_n +$ proved $C_2 \lor a$ from $H_1, \ldots, H_n$

Propositional resolution rule:

\[
\begin{array}{cccc}
H_1 & \cdots & H_n & H_1 & \cdots & H_n \\
\vdots & & \vdots & \vdots & & \vdots \\
C_1 \lor \bar{a} & & C_2 \lor a \\
\hline 
C
\end{array}
\]
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
$a_{i,j}$ means pigeon $i$ in hole $j$
Each pigeon in a hole: for all $i$,

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

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

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

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/](http://minisat.se/)
- Armin Biere’s solvers [http://fmv.jku.at/software/index.html](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 \( t \))

\[ a \lor \overline{b} \lor c \lor \overline{d} \]

in context \( a = f, \ b = t, \ d = t \), deduce \( c = t \).
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 $l := f$, scan associated list

- For each clause with one literal assigned $t$, ignore the clause
- For each clause with $\geq 1$ unassigned literal $l'$, move clause to the list for $l'$
- For each clause with 1 unassigned literal $l'$, $l' := t$
- 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

Introduction

Propositional logic
  Generalities
  Binary decision diagrams
  Satisfiability checking
  Applications

First-order logic

Applications

Beyond DPLL(T)
Equivalence checking

Two combinatorial circuits
How to check they compute the same?
Equivalence checking

Two combinatorial circuits
How to check they compute the same?

$(a_1, \ldots, a_n)$ output from $A$
$(b_1, \ldots, b_n)$ output from $B$ (same inputs)
assert $(a_1 \oplus b_1) \lor \cdots \lor (a_n \oplus b_n)$
Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants
Schedule

Introduction

Propositional logic

First-order logic
  Basics
  Decision in fixed theories
  DPLL(T)

Applications

Beyond DPLL(T)

Quantifier elimination
Quantifier-free first order formulas

e.g. \((P(f(x, z), y) \land R(y) \land T) \lor S(y, g(z, x))\)

Ordinary connectives \(\land, \lor, \neg\)...

**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: $\lt, \leq, \gt, \geq...$
$=$ 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

∀, ∃

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

∀x ((∀yP(x, y)) ∨ (∃yQ(x, y)))

in prenex form

∀x∀y₁∃y₂ (P(x, y₁) ∨ Q(x, y₂))
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 $P^\mathcal{M} : \mathcal{M}^n \rightarrow \mathcal{M}$

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

Introduction

Propositional logic

First-order logic
   Basics
   Decision in fixed theories
   DPLL(T)

Applications

Beyond DPLL(T)

Quantifier elimination
Models with fixed interpretation of certain predicates

\[ \forall x \exists y \ y > x \land y < x + z \]

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)
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

Introduction

Propositional logic

First-order logic
  Basics
  Decision in fixed theories
  DPLL(T)

Applications

Beyond DPLL(T)

Quantifier elimination
DPLL(T)

(Improper terminology, should be CDCL(T))

\((x \leq 0 \lor x + y \leq 0) \land y \geq 1 \land x \geq 1\)

↓ dictionary of theory literals

\((a \lor b) \land c \land d\)

Solve, get \((a, b, c, d) = (t, f, t, t)\).

But \(x \leq 0 \land x \geq 1\) is a contradiction!

Add **theory lemma** \(\bar{a} \lor \bar{d}\)

Solve, get \((a, b, c, d) = (f, t, t, t)\).

But \(x + y \leq 0 \land x \geq 1\) is a contradiction!

Add **theory lemma** \(\bar{b} \lor \bar{c} \lor \bar{d}\).

The problem is **unsatisfiable**.
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 \textit{simplex}. Extract from the tableau the contradictory subset of assignments.
LRA Example

\begin{align*}
2 & \leq 2x + y \\
-6 & \leq 2x - 3y \\
-1000 & \leq 2x + 3y & \leq 18 \\
-2 & \leq -2x + 5y \\
20 & \leq x + y.
\end{align*}
LRA Example

\[
\begin{align*}
  a &= 2x + y & 2 \leq a \\
  b &= 2x - 3y & -6 \leq b \\
  c &= 2x + 3y & -1000 \leq c \leq 18 \\
  d &= -2x + 5y & -2 \leq d \\
  e &= x + y & 20 \leq e.
\end{align*}
\]
LRA Example

Gauss-like pivoting until:

\[
\begin{align*}
  e &= \frac{7}{16}c - \frac{1}{16}d \\
  a &= \frac{3}{4}c - \frac{1}{4}d \\
  b &= \frac{1}{4}c - \frac{3}{4}d \\
  x &= \frac{5}{16}c - \frac{3}{16}d \\
  y &= \frac{1}{8}c + \frac{1}{8}d.
\end{align*}
\]
LRA Example

\[ e = \frac{7}{16}c - \frac{1}{16}d \]

But: \( c \leq 18 \) and \( d \geq -2 \), so \(-7/16c - 1/16d \leq 8\).

But we have \( e \geq 20 \), thus \textbf{no solution}.

Relevant original inequalities can be combined into an unsatisfiable one (thus the \textbf{theory lemma})

\[
\begin{align*}
\frac{7}{16} & (-2x - 3y) \geq -\frac{7}{16} \times 18 \\
\frac{1}{16} & (-2x + 5y) \geq -\frac{1}{16} \times 2 \\
1 & x + y \geq 20 \\
0 & 0 \geq 12
\end{align*}
\]
Linear integer arithmetic

Linear real arithmetic +
- branching: if LRA model $x = 4.3$, then $x \leq 4 \lor x \geq 5$
- (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, x₀, y₀) the function mapping

- x ≠ x₀ to f[x]
- x₀ to y₀.
Quantifiers

Show this formula is true:

\[(\forall i \ 0 \leq i < j \implies t[i] = 42) \implies (\forall i \ 0 \leq i \leq j \implies \text{update}(t, j, 0)[i] = 42)\] (5)

Equivalently, unsatisfiable:

\[0 \leq i_0 \leq j \land \text{update}(t, j, 0)[i_0] = 0 \land (\forall i \ 0 \leq i < j \implies t[i] = 0)\]
Prove unsatisfiable:

\[ 0 \leq i_0 \leq j \land \text{update}(t, j, 0)[i_0] = 0 \land (\forall i \ 0 \leq i < j \implies t[i] = 0) \]

By instantiation \( i = i_0 \):

\[ 0 \leq i_0 \leq j \land \text{update}(t, j, 0)[i_0] = 0 \land (0 \leq i_0 < j \implies t[i_0] = 0) \]

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

```c
/*@ 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 get_sign.c
$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\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
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;
}
$ 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 (00000000000000000000000001100000)
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 (00000000000000000000000001100000)
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

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants
Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

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

$b_i \text{ true } \leftrightarrow \text{ trace goes through } b_i$

Cost for the trace:
$\sum b_i \times cost_i$
Step 1: encode instructions
(Linear Integer Arithmetic)

Static Single Assignment form:
1 SMT variable ↔ 1 SSA variable

\[-10000 \leq x_{old.0} \leq 10000\]
\[-10000 \leq x.0 \leq 10000\]
\[add = (x_{old.0} + 10)\]
\[sub = (x_{old.0} - 10)\]
\[x.3 = (x_{old.0} - 10)\]
\[b_2 \Rightarrow (x.2 = \text{ite}(t_1, x.1, x.0))\]
\[b_4 \Rightarrow (x.1 = \text{ite}(t_3, x.3, x.2))\]
Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

1 SMT variable $\leftrightarrow$ 1 SSA variable

\[-10000 \leq x_{\text{old}.0} \leq 10000\]
\[\land\quad -10000 \leq x.0 \leq 10000\]
\[\land\quad \text{add} = (x_{\text{old}.0} + 10)\]
\[\land\quad x.1 = (x_{\text{old}.0} + 10)\]
\[\land\quad \text{sub} = (x_{\text{old}.0} - 10)\]
\[\land\quad x.3 = (x_{\text{old}.0} - 10)\]
\[\land\quad b_2 \Rightarrow (x.2 = \text{ite}(t_{1_2}, x.1, x.0))\]
\[\land\quad b_4 \Rightarrow (x.1 = \text{ite}(t_{3_4}, x.3, x.2))\]
Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:
1 SMT variable $\leftrightarrow$ 1 SSA variable

$-10000 \leq x_{old.0} \leq 10000$
$\land -10000 \leq x.0 \leq 10000$
$\land add = (x_{old.0} + 10)$
$\land x.1 = (x_{old.0} + 10)$
$\land sub = (x_{old.0} - 10)$
$\land x.3 = (x_{old.0} - 10)$
$\land b_2 \Rightarrow (x.2 = \text{ite}(t_1_2, x.1, x.0))$
$\land b_4 \Rightarrow (x.1 = \text{ite}(t_3_4, x.3, x.2))$
Step 2: encode control flow
(Very similar to ILP)

\[
\begin{align*}
& b_0 = b_4 = true \\
& b_1 = t_{0\_1} \\
& b_2 = (t_{0\_2} \lor t_{1\_2}) \\
& \vdots \\
& t_{0\_1} = (b_0 \land (x.0 > add)) \\
& \vdots \\
& t_{0\_2} = (b_0 \land (x.0 > add)) \\
& \vdots \\
& t_{2\_3} = (b_0 \land (x.0 > add)) \\
& \vdots \\
& t_{2\_4} = (b_0 \land (x.0 > add)) \\
& \vdots \\
& t_{3\_4} = (b_0 \land (x.0 > add)) \\
& \vdots \\
& x_{\text{old.}1} = \phi [x.3, \text{if.then4}], [x.2, \text{if.end6}] \\
\end{align*}
\]
Step 3: encode timings

\[
\begin{align*}
\land & \quad c_{0,1} = (\text{if}(t_{0,1}) \text{ then } 15 \text{ else } 0) \\
\land & \quad c_{0,2} = (\text{if}(t_{0,2}) \text{ then } 14 \text{ else } 0) \\
\land & \quad \vdots \\
\land & \quad \vdots \\
\land & \quad cost = (c_{0,1} + c_{0,2} + c_{1,2} + c_{2,3} + c_{2,4} + c_{3,4})
\end{align*}
\]
1 satisfying assignment

 ↔ 1 program trace:

\[ b_0 = b_1 = b_2 = b_4 = \text{true} \]
\[ b_3 = \text{false} \]
\[ t_0_1 = t_1_2 = t_2_4 = \text{true} \]
\[ t_0_2 = t_2_3 = t_3_4 = \text{false} \]
\[ x_{old.0} = 50 \]
\[ x.0 = 61 \]
\[ \text{add} = 60 \]
\[ x.1 = 60 \]
\[ x.2 = 60 \]
\[ \text{sub} = 40 \]
\[ \text{cost} = 32 \]
1 satisfying assignment

\[ 1 \text{ program trace:} \]

\[
\begin{align*}
  b_0 &= b_1 = b_2 = b_4 = \text{true} \\
  b_3 &= \text{false} \\
  t_{0_1} &= t_{1_2} = t_{2_4} = \text{true} \\
  t_{0_2} &= t_{2_3} = t_{3_4} = \text{false} \\
  x_{\text{old.0}} &= 50 \\
  x.0 &= 61 \\
  \text{add} &= 60 \\
  x.1 &= 60 \\
  x.2 &= 60 \\
  \text{sub} &= 40 \\
  \text{cost} &= 32
\end{align*}
\]

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 $\land l_i \implies \phi$, optimize locally in $\land l_i$, get a new bound.
Diamonds

Corresponds to sequence of \( n \) “if-then-else”:

\[
\begin{align*}
\text{if } (b[i]) & \{ \text{ timing 2 } \} \text{ else } \{ \text{ timing 3 } \} \\
\text{if } (b[i]) & \{ \text{ timing 3 } \} \text{ else } \{ \text{ timing 2 } \}
\end{align*}
\]

\( D(n) \) the unsatisfiable formula:

\[
\begin{align*}
\text{for } 0 \leq i < n & \begin{cases} \\
    x_i - t_i \leq 2 \\
y_i - t_i \leq 3 \\
(t_{i+1} - x_i \leq 3) \lor (t_{i+1} - y_i \leq 2)
\end{cases} \\
t_n - t_0 > 5n
\end{align*}
\]
Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)
  WCET example
  CDCL explosion
  Abstract CDCL (ACDCL)
  Model-construction satisfiability calculus (MCSAT)
Behavior of SMT-solvers

![Graph showing the behavior of SMT-solvers over time](image)

- Z3 3.2
- Z3 4.3.1
- MathSAT 5.2.6
- SMTInterpol
- \( \sim 2.22^n \)
Cost increases near bound

\[ Z3 3.2 \approx \frac{1}{2^{(B-90)}} \]
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

 delve into 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 $z = x - y \mapsto x \in [-9, +\infty)$

If too coarse, split intervals.
Akin to constraint programming.
Learning in ACDCL

Constraints \((y = x) \land (z = x \cdot y) \land (z \leq -1)\)
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)
In DPLL(T), assign only to Booleans and atoms from original formula.
In MCSAT, assign to propositional atoms \textit{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 $x_1, \ldots, x_n$ \textit{(partial projection)}. 
Example: diamonds

\[
\begin{aligned}
&\text{for } 0 \leq i \leq 2 \begin{cases} 
& x_i - t_i \leq 2 \\
& y_i - t_i \leq 3 \\
& t_{i+1} - x_i \leq 3 \lor t_{i+1} - y_i \leq 2 
\end{cases} \\
&t_0 = 0 \\
t_3 \geq 16
\end{aligned}
\]

Pick \( t_0 \mapsto 0, t_1 - x_0 \leq 3 \mapsto t, x_0 \mapsto 0, \)
\( t_1 \mapsto 0, t_2 - x_1 \leq 3 \mapsto t, x_1 \mapsto 0, \)
\( t_2 \mapsto 0, t_3 - x_2 \leq 3 \mapsto t, x_2 \mapsto 0. \)

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)

\[
\begin{align*}
t_3 - x_2 &\leq 3 \\
t_3 &\geq 16
\end{align*}
\implies x_2 \geq 13
\]

Possible to learn

\[ t_3 - x_2 > 3 \lor x_2 \geq 13 \]

Retract \( x_2 \mapsto 0 \).
Backtracking

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

\[
\begin{align*}
\{ & x_2 \geq 13 \quad x_2 - t_2 \leq 2 \quad \implies \quad t_2 \geq 11 \\
\end{align*}
\]

Thus learn

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

t_3 - x_2 \leq 3 \iff t$ retracted.
Continuation

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

\[
t_3 - x_2 \leq 3 \lor t_2 \geq 11
\]

Thus

\[
\begin{cases}
  t_3 - x_2 > 3 \lor t_2 \geq 11 \\
t_3 - x_2 \leq 3 \lor t_2 \geq 11
\end{cases} \implies t_2 \geq 11
\]

One learns \( t_2 \geq 11 \).

Then \( t_1 \geq 6 \) similarly.

But then no satisfying assignment to \( t_0 \)!
(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

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Interpolants
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

\[ \implies \text{decidability} \]
Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Booleans

Projecting conjunctions

Substitution methods
Resolution

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

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

For clauses where $a$ appears, apply resolution:

$$
C_i' \lor a \quad C_j' \lor \neg a \\
\frac{C_j' \lor C_j'}{C_i' \lor C_j'}
$$

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

$$
C_1' \land \cdots \land C_{n'}' \equiv \exists a \ (C_1 \land \cdots \land C_n)
$$
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 $a \lor b \lor c$ in addition to $a \lor b$.
- Eliminate all variables: obtain the empty clause ($\mathbf{f}$) 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

\[ \exists x \ F(x) \equiv F(0) \lor F(1) \]
\[ \forall x \ F(x) \equiv F(0) \land F(1) \]

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

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

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

\[ C \text{ conjunction of linear inequalities.} \]

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

Closed convex polyhedron in \( x, y, \ldots, \epsilon \).

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 \text{ vertices} + 2 \text{ rays}
Chernikova’s algorithm

Often applied following Le Verge’s remarks.
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 $x \leq \frac{1}{n}(b - \ldots)$
- $L_-$, where $x$ has negative coefficient $((-n).x + \cdots \leq b)$, thus $x \geq \frac{1}{n}(b - \ldots)$
- $L_0$, without $x$

Otherwise said

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

\[\exists x \max_i l_i^- (y, \ldots) \leq x \leq \min_j l_j^+ (y, \ldots)\]
iff \[\max_i l_i^- (y, \ldots) \leq \min_j l_j^+ (y, \ldots)\]
iff for all \(i\) and \(j\) \(l_i^- (y, \ldots) \leq l_j^+ (y, \ldots)\)

Elimination:

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

Eliminate $y$ from:

$x + y \leq 1 \land -x + y \leq 1 \land x - y \leq 1 \land -x - y \leq 1$

\[x + y \leq 1 \land x - y \leq 1 \implies x \leq 2 \equiv x \leq 1\]
\[-x + y \leq 1 \land -x - y \leq 1 \implies 0 \leq 2\]
\[-x + y \leq 1 \land x - y \leq 1 \implies 0 \leq 2\]
\[-x + y \leq 1 \land -x - y \leq 1 \implies -2x \leq 2 \equiv x \geq -1\]

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 $n^{2p}$.

But McMullen’s bound (# dimension-$k$ faces of a polyhedron with $\nu$ 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
  \[ a_1 x_1 + \cdots + a_n x_n \leq B' \text{ with } B' \leq B \]

- **Linear programming**: if we have \( C \) and add \( C' \)
  \( (a_1 x_1 + \cdots + a_n x_n \leq B) \),
  - test emptiness of \( C \land \neg C' \) 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](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

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination
  Booleans
  Projecting conjunctions
  Substitution methods
Basic ideas of substitution methods

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

**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 « \( \forall x \exists y \ x = n \cdot x \) » for any natural \( n \).
Test points
Ferrante & Rackoff’s method

\[ \exists x \ F \text{ is true iff} \]

\[ \begin{align*}
\text{true for } x \rightarrow -\infty, \ x \rightarrow +\infty, \\
\text{true for all } x \text{ intersection point or median to intersections}
\end{align*} \]

Let \( x_1, \ldots, x_n \) the intersections as functions of \( y, z, \ldots \)

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

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 $x \to -\infty$
- true for all $x = x_i$ given $x \geq x_i(y, \ldots)$
- true for all $x = x_i + \varepsilon$ given by $x > x_i(y \text{ dots})$

$\varepsilon$ infinitesimal, $x \geq t + \varepsilon$ means $x > t$

\[
\exists x \ F \equiv F[x \mapsto -\infty] \lor F[x \mapsto +\infty] \lor \bigvee_{i} F[x \mapsto x_i] \lor \bigvee_{j} F[x \mapsto x'_i + \varepsilon]
\]

**Linear** number of substitutions.
Schedule

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination

Booleans

Projecting conjunctions

Substitution methods
Presburger arithmetic

+, −, ≥ (+multiplication by constant)

Does it admit quantifier elimination?
Presburger arithmetic

\[ +, -, \geq \text{ (+multiplication by constant)} \]

Does it admit quantifier elimination?

No: \[ \exists x \; y = 2x \not\equiv \text{ a quantifier-free formula} \]

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 \( \delta \) the least common multiple of \( n \) in \( n \mid x' + t \) or \( n \nmid x' + t \). Divisibility predicates have \( \delta \)-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 $t$, $x' > t$ by $f$.
Obtain $F_{-\infty}$. Only divisibility atoms.

The problem is now $\delta$-periodic, thus take

$$F'[x' \mapsto 1] \lor \cdots \lor F'[x' \mapsto \delta]$$
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 \( \delta \)-periodicity:

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

Finally:

\[
\exists x \ F \equiv \exists x' \ F' \equiv \bigvee_{j=1}^{\delta} F'_{-\infty}[x' \mapsto j] \lor \bigvee_{j=1}^{\delta} \bigvee_{t \in B} F'[x' \mapsto b + j]
\]
Some optimizations

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

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

Eliminate in $\exists y_1 \ldots y_n F'[x' + j]$ 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 $\mathbb{N}$
$\leadsto$ finite automaton recognizing $n$-tuples of binary words
$\leadsto$ finite automaton recognizing words over $\{0, 1\}^n$

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

\[
\begin{align*}
q_0 & \quad \langle 1, 1, 0 \rangle \quad q_1 \\
\langle 0, 0, 0 \rangle & \quad \langle 0, 1, 1 \rangle \\
\langle 1, 0, 1 \rangle & \quad \langle 1, 1, 1 \rangle \\
\langle 0, 1, 0 \rangle & \quad \langle 1, 0, 0 \rangle \\
\langle 0, 1, 0 \rangle & \quad \langle 0, 1, 0 \rangle \\
\langle 1, 1, 0 \rangle & \quad \langle 0, 1, 1 \rangle \\
\langle 0, 0, 1 \rangle & \quad \langle 0, 0, 1 \rangle \\
\langle 1, 0, 0 \rangle & \quad \langle 1, 0, 0 \rangle \\
\langle 1, 1, 1 \rangle & \quad \langle 1, 1, 1 \rangle
\end{align*}
\]
Constructions

- Constants: easy.
- \( \mathbb{Z} \): use a sign bit, or encode \( \geq 0 \) into even numbers and \( < 0 \) into odd numbers.
- Successor, addition etc.: propagate \textit{carry} inside the automaton state.
- \( \land \): intersection of regular languages
- \( \lor \): 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 \( dP/dX \)
- extraction of leading coefficient: from \( \sum_{k=0}^{d} a_k X^k \) \((a_k \in \mathbb{Z}[Y_1, \ldots, Y_m] \setminus \{0\})\) 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 \( A, B \in \mathbb{Z}[X, Y_1, \ldots, Y_m] \), \( \deg A \geq \deg B \), and \( D \) leading coefficient of \( B \), there exist unique \( Q \) and \( R \) s.t. \( D^{\deg A - \deg B + 1} \cdot A = QB + R \); 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 $]-\infty, \gamma_1[, \{\gamma_1\}, ]\gamma_1, \gamma_2[, \{\gamma_2\}, \ldots$ give a sign ($-, 0, +$).

<table>
<thead>
<tr>
<th></th>
<th>$-\infty$</th>
<th>$\gamma_1$</th>
<th>$\gamma_2$</th>
<th>$\gamma_3$</th>
<th>$+\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$c$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>$4c - b^2$</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$2x + b$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$0$</td>
<td>$+$</td>
</tr>
<tr>
<td>$x^2 + bx + c$</td>
<td>$+$</td>
<td>$0$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Idea of the proof

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

<table>
<thead>
<tr>
<th></th>
<th>$b$</th>
<th>$c$</th>
<th>$4c - b^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>
The other polynomials

The sign of its derivative imposes the behavior $2x + b$, thus a root $\gamma_2$

<table>
<thead>
<tr>
<th></th>
<th>$-\infty$</th>
<th>$\gamma_2$</th>
<th>$+\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$c$</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$4c - b^2$</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$2x + b$</td>
<td>−</td>
<td>0</td>
<td>+</td>
</tr>
</tbody>
</table>
From one stratum to one higher stratum

\[ x^2 + bx + c \text{ is } + \text{ when } x \to \pm \infty \]

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

| \( \begin{array}{c}
 b \\
 c \\
 4c - b^2 \\
 2x + b \\
 x^2 + bx + c \\
\end{array} \) | \(-\infty\) | \(\gamma_2\) | \(+\infty\) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( c )</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( 4c - b^2 )</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( 2x + b )</td>
<td>-</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>( x^2 + bx + c )</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
Adding other roots

By continuity, need extra roots $\gamma_1$ and $\gamma_3$

<table>
<thead>
<tr>
<th></th>
<th>$-\infty$</th>
<th>$\gamma_1$</th>
<th>$\gamma_2$</th>
<th>$\gamma_3$</th>
<th>$+\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$c$</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>$4c - b^2$</td>
<td>−</td>
<td>+</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>$2x + b$</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>$x^2 + bx + c$</td>
<td>+</td>
<td>0</td>
<td>−</td>
<td>0</td>
<td>+</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th></th>
<th>(-\infty)</th>
<th>(\gamma_1)</th>
<th>(\gamma_2)</th>
<th>(\gamma_3)</th>
<th>(+\infty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>(c)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>(4c - b^2)</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(2x + b)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>(x^2 + bx + c)</td>
<td>+</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
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

Introduction

Propositional logic

First-order logic

Applications

Beyond DPLL(T)

Quantifier elimination
  Booleans
  Projecting conjunctions
  Substitution methods
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

B

ok

fail

\begin{align*}
i' &= i \\
j' &= j \\
i &\geq 100 \\
j &< 210
\end{align*}

\begin{align*}
i' &= 0 \\
j' &= 0 \\
i &< 100
\end{align*}

\begin{align*}
i' &= i + 1 \\
j' &= j + 2 \\
i &\geq 100 \\
j &\geq 210
\end{align*}
A bad counterexample

Try to find values for the red path:
\[ i_1 = 0 \land j_1 = 0 \land i_2 = i_1 \land j_2 = j_1 \land i \geq 100 \land j \geq 210 \]
A bad counterexample

Try to find values for the red path:

\[ i_1 = 0 \land j_1 = 0 \land i_2 = i_1 \land j_2 = j_1 \land i \geq 100 \land j \geq 210 \]

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.
Two step explanation for infeasible path:

1. $i_1 = 0 \land j_1 = 0 \land i_2 = i_1 \land j_2 = j_1 \Rightarrow j_2 = 2i_2 \land i \geq 100$

2. $j_2 = 2i_2 \land i_2 \geq 100 \Rightarrow j_2 < 210$
A refinement

Two step explanation for infeasible path:

1. \( i_1 = 0 \wedge j_1 = 0 \wedge i_2 = i_1 \wedge j_2 = j_1 \Rightarrow j_2 = 2i_2 \wedge i \geq 100 \)
2. \( j_2 = 2i_2 \wedge i_2 \geq 100 \Rightarrow j_2 < 210 \)

This is a Craig interpolant.
A good refinement

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

\[ j = 2i \wedge i \leq 100 \]

\[ i' = i \]
\[ j' = j \]
\[ i \geq 100 \]
\[ j < 210 \]

\[ i' = 0 \]
\[ j' = 0 \]

A

\[ i < 100 \]
\[ i' = i + 1 \]
\[ j' = j + 2 \]
Another refinement

Two step explanation for infeasible path:

1. \( i_1 = 0 \land j_1 = 0 \land i_2 = i_1 \land j_2 = j_1 \Rightarrow i_2 = 0 \land j_2 = 0 \)
2. \( i_2 = 0 \land j_2 = 0 \Rightarrow j_2 < 210 \)

This is another Craig interpolant.
Another refinement

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

```
i' = i
j' = j
i ≥ 100
j < 210
```

```
i' = i
j' = j
i ≥ 100
j ≥ 210
```

A

B

ok

fail

$$\forall x (x^2 + 2 > 0)$$
Further refinement

Two step explanation for infeasible path:

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

This is another Craig interpolant.
Further refinement

```c
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 !)
Suppose $A \land B$ unsatisfiable (aka $A \implies \neg B$)
Obtain a resolution proof of $f$, 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)

\[
\begin{align*}
\begin{array}{c}
c \quad [c \downarrow B] \\
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
c \quad c \in A \\
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
c \quad [t] \\
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
c \quad c \in B \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{c}
\begin{array}{c}
v \lor c \quad [l_1] \\
\end{array}
\end{array} \\
\begin{array}{c}
\begin{array}{c}
\bar{v} \lor d \\
\end{array}
\end{array} [l_2]
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
c \lor d \quad [l_1 \lor l_2]
\end{array}
\end{align*}
\]

\[v \text{ does not occur in } B\]

\[
\begin{align*}
\begin{array}{c}
\begin{array}{c}
v \lor c \quad [l_1] \\
\end{array} \\
\begin{array}{c}
\begin{array}{c}
\bar{v} \lor d \\
\end{array}
\end{array} [l_2]
\end{array}
\end{align*}
\begin{align*}
\begin{array}{c}
c \lor d \quad [l_1 \land l_2]
\end{array}
\end{align*}
\]

\[v \text{ occurs in } B\]
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 $c$  

\[ A \models l_c \lor (c \setminus B) \]
\[ B, l_c \models c \downarrow B \]
\[ l_c \preceq A \text{ and } l_C \preceq B \]

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 \( C_1 \land \cdots \land C_n \)
\( C_i \) 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: \( A' \models l_\phi, B' \models \neg l_\phi \)
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).
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

⇒ **brittleness**

Search for “simpler”, more “beautiful” interpolants?
An interpolation problem

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

\[
A_1 = x \leq 1 \land y \leq 4 \\
A_2 = x \leq 4 \land y \leq 1 \\
A = A_1 \lor A_2 \\
B = x \geq 3 \land y \geq 3.
\]

(6)

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