Satisfiability Modulo Theories in an Undergraduate Class

Yoni Zohar

Bar-Ilan University, Israel

ThEdu'2022 @ Haifa

Outline

1. Introduction
2. Propositional Logic
3. Uninterpreted Functions
4. Predicates and Arithmetic
5. Bit-vectors and Sorts
6. Arrays and Quantifiers

General Info

- 13 Lectures
- 90 minutes each
- Whiteboard and demonstrations (no slides)
- 12 students
- No Logic Background
- Hybrid – 50% on Zoom and 50% in class
- Some undergrads, some graduate
- 3 HW
- Final Exam

Goals

- Attract graduate students
- Advertise automated reasoning
- Adding it to their tool-box
- Expose to new ways of thinking:
  - Encoding instead of coding
  - Calculi instead of algorithms
  - Formal and full proofs
  - logic



Resources

Books

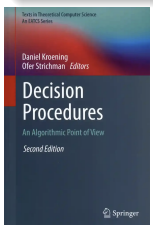
- Decision Procedures (Kroening & Strichman 2016)
- The Calculus of Computation (Bradely & Manna 2017)
- A Mathematical Introduction to Logic (Enderton 2001)

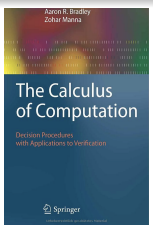
Articles

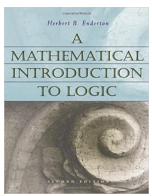
Solving SAT and SAT Modulo Theories (Nieuwenhuis, Oliveras, Tinelli 2006)

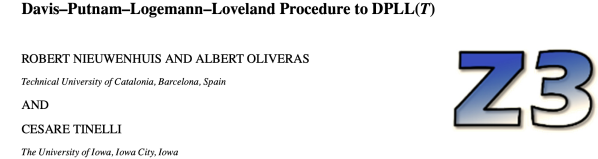
Tools

- z3
- pysmt











Guidlines To Self

- Hands-on
  - Open the terminal in every class
  - HW should include coding
- Depth over width
  - Formally define everything
  - Prove (almost) everything
  - Price: Less topics, less generality, more simplifications, less depth?
- Be Lazy
  - Plan each subject from the end
  - Only include what is really necessary
  - Price: some repetitions
- Active Learning
  - Each class should include 3-4 minutes of active learning
  - Tasks can be impossible
- Include Many Examples
  - At least one example after every definition
  - Example for each algorithm

General Structure

- A real problem
- Logic can solve it
- We don’t have a rich-enough framework
- Introduce required definitions and theory
- Decision Procedures
  - Implementation



Outline

2 Propositional Logic
- Motivation
- Theory
- Solving

Motivation

A Motivating Example

```
def foo(a, b):
if not a and not b:
  h()
else:
  if not a:
    g()
  else:
    f()
```

```
def goo(a, b):
  if a:
    f()
  else:
    if b:
      g()
    else:
      h()
```

Are `foo` and `goo` equivalent?

Based on Kroening & Strichman 2016

From Programs to Propositional Logic

- Boolean variables: a, b, f, g, h
- $\varphi_{foo} = \text{ite}(\neg a \wedge \neg b, h, \text{ite}(\neg a, g, f))$
- $\varphi_{goo} = \text{ite}(a, f, \text{ite}(b, g, h))$
- $\text{ite}(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
- Is The formula $\varphi_{foo} \leftrightarrow \varphi_{\psi}$ valid?

```
verification.cnf
brute_force_solver
```

Outline

2 Propositional Logic
- Motivation
- Theory
- Solving

Definitions

The Real Deal

- variables, formulas, valuations
- Satisfaction
- Satisfiability, Validity, Contratiction



CNF

- SAT solvers do not parse every possible formula
- They typically parse CNF
- Is this a problem? Our formula $\varphi_{\mathrm{foo}} \leftrightarrow \varphi_{\mathrm{goo}}$ is not in CNF.

Definition of CNF

- Literals
- Clauses
- CNF

Translation to CNF

Using Equivalences

- Definition of equivalence
- Theorem: for every formula there is an equivalent CNF formula
- Proof sketch: use the following equivalences:
  - $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
  - $(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$
  - ...

Bad News

- B from the theorem might be exponential
- Parsable by a SAT-solver but impracticle
- So we are still in a problem with $\varphi_{\mathrm{foo}} \leftrightarrow \varphi_{\mathrm{goo}}$

Solution

- Do we really need A and B to be equivalent?
- Enough for them to be equi-SAT

Tseitin's Transformation

Theorem

For every formula there exists an equis-sat CNF whose size is linear.

Proof Structure

- Number of subformulas is linear
- Let A be a formula. For every subformula C introduce a variable p_C.
- $B := p_A \wedge \bigwedge_{C \in sub(A)} E(C)$

$E(C)$

$$E(C) = \begin{cases} CNF(p_C \leftrightarrow C) & C \text{ is variable} \\ CNF(p_C \leftrightarrow true) & C \text{ is true} \\ CNF(p_C \leftrightarrow false) & C \text{ is false} \\ CNF(p_C \leftrightarrow \neg p_D) & C = \neg D \\ CNF(p_C \leftrightarrow (p_{C_1} \wedge p_{C_2})) & C = C_1 \wedge C_2 \\ CNF(p_C \leftrightarrow (p_{C_1} \vee p_{C_2})) & C = C_1 \vee C_2 \\ CNF(p_C \leftrightarrow (p_{C_1} \rightarrow p_{C_2})) & C = C_1 \rightarrow C_2 \\ CNF(p_C \leftrightarrow (p_{C_1} \leftrightarrow p_{C_2})) & C = C_1 \leftrightarrow C_2 \end{cases}$$

Tseitin's Transformation

Theorem

For every formula there exists an equis-sat CNF whose size is linear.

Proof Structure

- Number of subformulas is linear
- Let A be a formula. For every subformula C introduce a variable p_C.
- $B := p_A \wedge \bigwedge_{C \in sub(A)} E(C)$

$E(C)$

$$CNF(p \leftrightarrow C) = \begin{cases} (\neg p_C \vee C) \wedge (\neg C \vee p_C) & C \text{ is variable} \\ (\neg p_C \vee true) \wedge (false \vee p_C) & C \text{ is true} \\ (\neg p_C \vee false) \wedge (true \vee p_C) & C \text{ is false} \\ (\neg p_C \vee \neg P_D) \wedge (p_D \vee p_C) & C \text{ is } \neg D \\ (\neg p_C \vee p_{C_1}) \wedge (\neg p_C \vee p_{C_2}) \wedge (\neg p_{C_1} \vee \neg p_{C_2} \vee p_C) & C \text{ is } C_1 \wedge C_2 \\ (\neg p_C \vee p_{C_1} \vee p_{C_2}) \wedge (\neg p_{C_1} \vee p_C) \wedge (\neg p_{C_2} \vee p_C) & C \text{ is } C_1 \vee C_2 \\ (\neg p_{\{C\}} \vee \neg p_{C_1} \vee p_{C_2}) \wedge (p_{C_1} \vee p_C) \wedge (\neg p_{C_2} \vee p_C) & C \text{ is } C_1 \rightarrow C_2 \\ (\neg p_C \vee \neg p_{C_1} \vee p_{C_2}) \wedge (\neg p_C \vee p_{C_1} \vee \neg p_{C_2}) \wedge & C \text{ is } C_1 \leftrightarrow C_2 \\ (p_C \vee \neg p_{C_1} \vee \neg p_{C_2}) \wedge (p_C \vee p_{C_1} \vee p_{C_2}) & \end{cases}$$

Tseitin (cont.)

Proof

- Linear # of new variables, constant number of clauses for each.
- Key idea: using definition variables in more complex formulas
- Correctness 1: Given $v \models A$, extend v to v' with $v'(p_C) := C$.
- Correctness 2: Given $v \models B$, prove that $v(p_C) = v(C)$ for each C.



Using SAT-solvers

Representing Boolean Formulas in a Computer

- Independent Study
- Come up with a format
- Gradually came to DIMACS

Satisfiability and Validity

- Must formulate as satisfiability
- Theorem: φ is valid iff $\neg\varphi$ is SAT

```
p cnf 22 49
6 2 0
17 -18 0
-20 21 -13 0
14 -10 -4 0
3 -19 0
10 -14 0
-12 13 0
```

Polynomial SAT

Particular Instances of SAT

- 2-SAT, Horn-SAT, dual-Horn-SAT
- Proof for Horn-SAT
  - $O(n^2)$ algorithm:
  - For each unit clause, delete its negation from every clause and delete every clause that includes it.
  - Return SAT if no empty clause was reached.
  - Proved complexity, correctness
  - Did not mention:
    - Watched Literals
    - Linear time algorithm

Students' Reaction So Far

- Amazed (by everything: Horn and non-Horn)
- We take for granted
- They don't
- But then...

Outline

2 Propositional Logic
- Motivation
- Theory
- Solving

Solving SAT

Book vs. Paper

- Book: pseudo-code, efficient, several techniques
- Paper: calculus, scheme, abstract

DPLL

- Calculus from Nieuwenhuis, Oliveras, Tinelli 2006
- Simple, ”classic” DPLL
- Active learning: try to solve a formula and tell me what you did
- Identified steps and formulated them as the DPLL rules.
- General idea: trying to construct a valuation

Definitions

- Configuration: *Fail* or (M, F, D) where:
  - M is a list of literals
  - F is a CNF formulas
  - D is a set of literals that occur in M

Calculus

The Rules

- Decide: $\frac{(M, F, D)}{(M\ell, F, \{\ell\})}$ where $var(\ell)$ occurs in F but not in M.
- Fail: $\frac{(M, F, D)}{Fail}$ where $D = \varnothing$ and $v_M \models \neg C$ for some $C \in F$
- BT: $\frac{(M\ell N, F, D)}{(M\bar{\ell}, F, D \backslash \{\ell\})}$ where $\ell \in D$, $N \cap D = \varnothing$, $\exists C \in F. v_{M\ell N} \models \neg C$.
- UP: $\frac{(M, F, D)}{(M\ell, F, D)}$ $\exists C. C \cup \{\ell\} \in F$, $var(\ell)$ not in M and $v_M \models \neg C$.

Algorithm

- Set initial configuration $([], F, \varnothing)$
- Apply rules as much as possible
- If *Fail* was derived, return “unsat”. Otherwise, return “sat”.

Not really an algorithm – an algorithm scheme. Python Implementation

DPLL (cont.)

Proof

- termination = No infinite derivation
- Full correctness

Did Not Cover

- backjumping / conflict analysis
- Which variable to decide on and how
- Data structures
- ...

Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T)

ROBERT NIEUWENHUIS AND ALBERT OLIVERAS

Technical University of Catalonia, Barcelona, Spain

AND

CESARE TINELLI

The University of Iowa, Iowa City, Iowa

HW 1

- Implement naive SAT solver
- Implement DPLL SAT solver
- I provided some test cases
- Graded on a larger set
  - Results: Almost all had a bug
- Prove that two programs are equivalent
  - Translate to logic
  - Transform to CNF
  - Transform to DIMACS
- Compare running times between the two solvers
- Prove some missing lemmas from class

Outline

3 Uninterpreted Functions
- Motivation
- Theory
- Solving Cubes
- Solving Generally

Into To FOL

A Motivating Example

```
def g(ig):
  og = ig
  for i in range(0,2):
    og = f(og, ig)
  return og
```

```
def h(ih):
  oh = f(f(ih, inh), in
  return oh
```

Are `foo` and `goo` equivalent?

Based on Kroening & Strichman 2016

From Programs to Logic

- $\psi_g : og1 = ig1 \wedge og2 = f(og1, ig1) \wedge og3 = f(og2, ig1)$
- $\psi_h : oh1 = f(f(ih1, ih1), ih1)$
- Is The formula $(ig1 = ih1 \wedge \psi_g \wedge \psi_h) \rightarrow og3 = oh1$ valid?

Demo `uf.smt2`

Outline

3. Uninterpreted Functions
   - Motivation
   - Theory
   - Solving Cubes
   - Solving Generally

Class 5

FOL

- Goal: analyze the equivalence formula
- Mono-sorted, no predicate symbols
- equality is a logical symbol
- Variables, Function Symbols, terms, formulas
- Semantics: normal equality
- Atomic formulas = equalities
- Validity, Satisfiability
- No Quantifiers

Flattenning Lemma

- Cube: Conjunction of literals
- Every cube is equi-satisfiable to a flat cube

Outline

3 Uninterpreted Functions
- Motivation
- Theory
- Solving Cubes
- Solving Generally

Class 6

Congruence Closure

- Solving cubes (conjunctions of literals)
- Active Learning: Is a given formula SAT?
- Identifying what we did, abstracting to rules

Calculus

- Configuration: (M, F) s.t.:
  - F is a flat cube
  - M is a set of sets of terms of F
- Top-level: $\frac{(M, F)}{(M \backslash \{X, Y\} \cup \{X \cup Y\}, F)}$ if $X, Y \in M$ and for some $s = t \in F$ we have $t \in X$ and $s \in Y$
- Congruence: $\frac{(M, F)}{(M \backslash \{X, Y\} \cup \{X \cup Y\}, F)}$ if $X, Y \in M$ and for some $Z \in M$ and $z, z' \in Z$, $f(z) \in X$ and $f(z') \in Y$
- Fail: $\frac{(M, F)}{\mathit{Fail}}$ if for some $X \in M$ and $t, s \in X$, $t \neq s \in F$

Correctness

Towards Correctness of CC

- Active learning: Given a saturated (M, F) – find a model for F
- Eventually got to the idea that M represents a model
- Brainstormed how to formalize this

CC Correctness

- Formal definition of M as a model:
  - Domain: M
  - variables: the set in which you are in
  - functions: Choose a repr., go to the set of the function application
- Termination: M decreases at every step

Herbrand Universe

- Originally thought: no soundness and completeness
- In practice: essential ingredient

Representation in Computer

SMTLIB and PySMT

- review of SMTLIB for UF: `smtlib_example.smt2`
- Review of simple PySMT: `pysmt_example.py`
- Review of *CC*: `https://github.com/yoni206/ar-class-2022-hw2/blob/main/uf_solver.py`

HW 2

- Implement CC
- I provided some helper functions
- Prove two functions are equivalent (encoding, running)
- Some missing lemmas and examples from class

CC (cont.)

Did not include

- $n \cdot log(n)$ algorithm for CC
- Union find data structure

Non-flat cubes

- Flatten
- Apply CC

Non-cubes

- DNF
- But not efficient

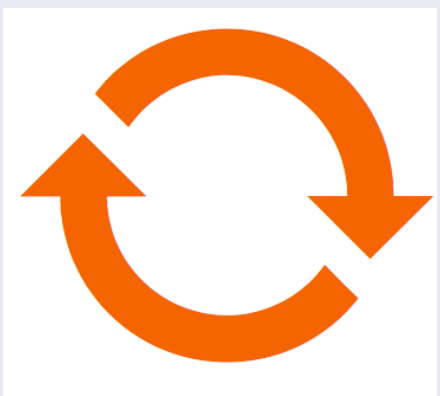
Outline

3 Uninterpreted Functions
- Motivation
- Theory
- Solving Cubes
- Solving Generally

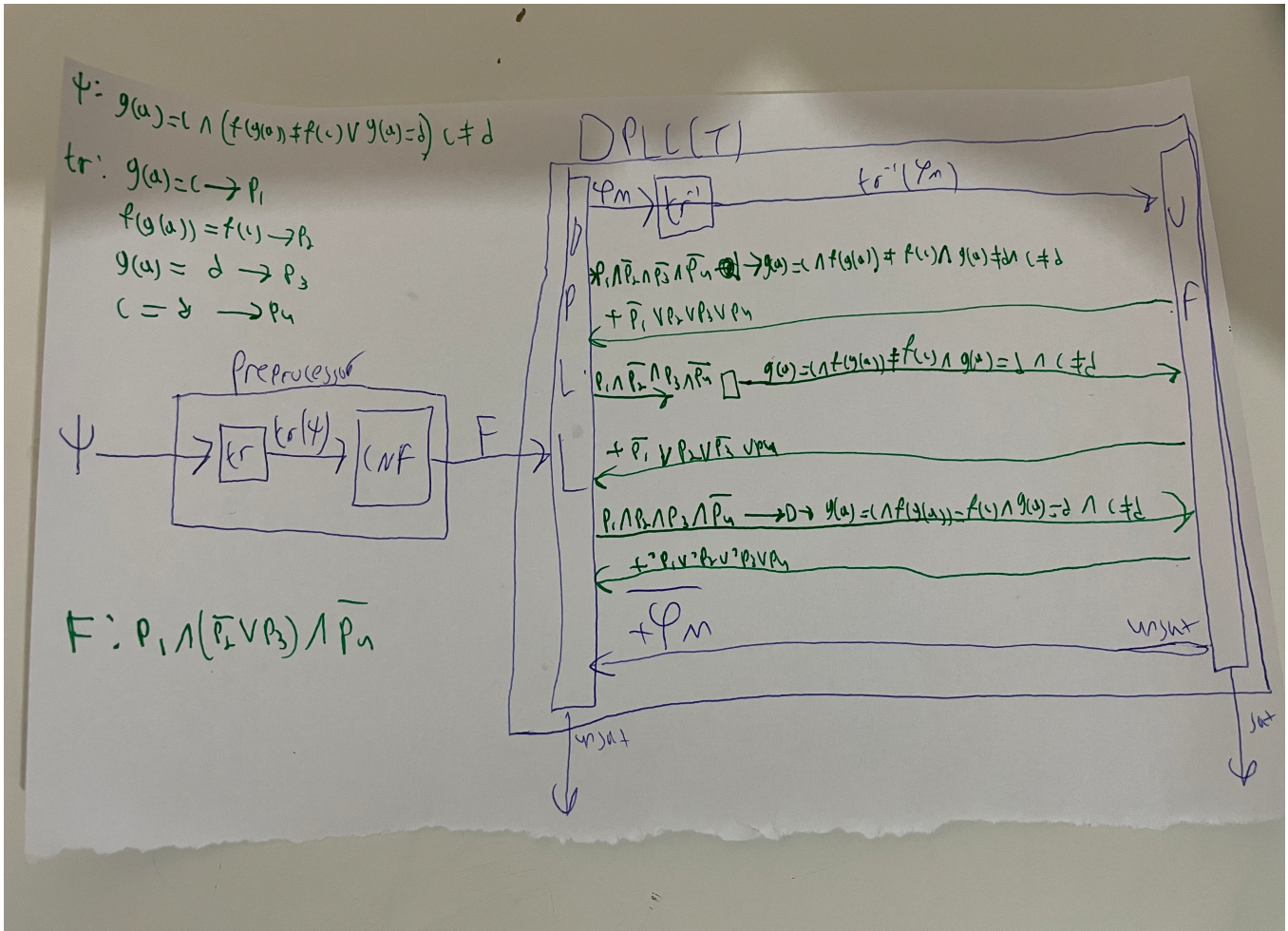
Towards DPLL(UF)

Task

- Try to solve a formula
- Try to use CC and DPLL
- Reflecting: we tried a Boolean assignment to the skeleton
- But it is not satisfiable in UF
- Add it's negation as a clause to block it
- repeat



DPLL(UF)



Calculus

Extension of DPLL

- *DPLL*
- New *UF* rule: $\frac{(M, F, D)}{([], F \wedge \overline{\varphi_M}, \varnothing)}$ if $tr^{-1}(\varphi_M)$ is not SAT in UF
- *tr* is 1-1 from FOL to Boolean formulas

Correctness

- Termination: Based on DPLL + finite numebr of valuations
- Correctness: Carefull translations using *tr*

Did not do

- Nothing but the naive DPLL(T) scheme
- Splitting on Demand
- Eager calls to theory solver
- ...

Outline

4 Predicates and Arithmetic
- Motivation
- Theory

Predicates, Arithmetic

Motivation for Predicates

- Active learning:
  - Define a signature and a structure for integers with $=$
  - Define a signature and structure for integers with $=$ and $<$
- Impossible, need a new construct

$$\mathbb{Z}$$

Outline

4 Predicates and Arithmetic
- Motivation
- Theory

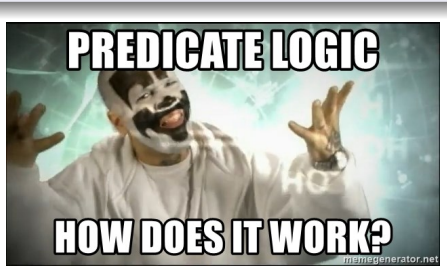
Predicates

Adding Predicates

- Predicates are similar to functions (have arities)
- New atomic formulas
- Extended interpretations – relations
- Another clause for satisfaction
- Extending the flattenning algorithm
- Repetition: good or bad?

Revisiting Integer Exapmle

$<$ is a predicate symbol.



Arithmetic

Linear Arithmetic Signature

- Signature: $\leqslant, +, -, 0, 1$
- Active learning: is $x < x$ is SAT? `arith_uf.smt2`, `arith.smt2`

Linear Arithmetic Structures

- *LRA*-structure: domain is $\mathbb{R}$ and interpretations as expected
- *LQA*-structure: domain is $\mathbb{Q}$ and interpretations as expected
- *LIA*-structure: domain is $\mathbb{Z}$ and interpretations as expected

Surprising Theorem

- φ is SAT by a *LRA*-structrue iff it is SAT by a *LQA*-structure.

Decision Procedures and More

- Simplex (did not show, studies elsewhere)
- complexity
- Non-linear

HW 3

Goal

- Experience in using SAT/SMT solvers
- Learning how to encode
- Placing the experience at the back of their heads

Description

`https://www.cs.tau.ac.il/~msagiv/courses/asv/z3py/guide-examples.htm`

HW

- Solve the install problem with PySMT
- Extend flattening for predicates
- UF vs. LIA
- SAT / UNSAT for formulas

Outline

5 Bit-vectors and Sorts
- Motivation
- Theory
- Solving

Bit-vectors

Motivation

- Example from Kroening & Strichman 2016
- `bv.cpp`, `bv_int.smt2`, `bv_many_sorted.cpp`
- Shows the problem of having only one domain in interpretations
- Motivation for many-sorted logic



Outline

5 Bit-vectors and Sorts
- Motivation
- Theory
- Solving

Sorts

Introducing Sorts

- Sorts
- New arities for function and predicates
- Variables are partitioned
- Terms, formulas must respect sorts
- Semantics: several domains, interpretations respect them
- Many examples



Bit-vectors

Signature

- Infinitely many sorts
- $<_n$ for each n
- $+_n$ for each n
- ...
- constants

Semantics

- Integer interpretations
- BV-structure – as expected, using integer interpretations

Demo

- Back to the original example
- bv_bv.smt2

Outline

5 Bit-vectors and Sorts
- Motivation
- Theory
- Solving

Decision Procedure

Bit-blasting

- Formal definition of translation from bit-vectors to Booleans
- Explicitely showed bit-wise and
- Briefly described addition
- Skipped the rest



Outline

6 Arrays and Quantifiers
- Motivation
- Theory
- Solving

Towards Arrays

```
def swap(a, i, j):
  tmp = a[i]
  a[i] = a[j]
  a[j] = tmp
```

Prove that if $a[i] = a[j]$ then the function does not change a.

Discussion

- Struggling
- Can't update a function
- We need Arrays
- For that, we need quantifiers

Outline

6 Arrays and Quantifiers
- Motivation
- Theory
- Solving

Quantifiers and Arrays

Adding Quantifiers

- Formulas: $\forall x.\varphi$, $\exists x.\varphi$
- Semantics: x-variants
- Examples

Arrays

- Sorts: I, E, A
- Symbols: r, w
- Semantics: Array-structure is a structure that satifies the arrays axioms
  - $\forall a, i, e.r(w(a, i, e), i) = e$
  - $\forall a, i, h, e.i \neq j \rightarrow r(w(a, i, e), j) = r(a, j)$
  - $\forall a, b.a \neq b \rightarrow \exists i.r(a, i) \neq r(a, i)$

Original Problem

Using arrays: `arrays.smt2`

Outline

6 Arrays and Quantifiers
- Motivation
- Theory
- Solving

Solving Arrays

Decision Procedure for a Fragment

- Formulas that do not equate arrays
- Procedure: reduction to UF
- Using a calculus
- Proof of correctness and termination

Did Not Show

- Full fragment
- Array property fragment
- Weak-equivalence

Regrets

Doing

- Termination of DPLL
  - Almost an entire class
  - For other systems it is easier
- Flattenning lemma
  - Took a lot of time
  - Not needed, simplifies only a bit

Not Doing

- Theory combination
- More on quantifiers
- More realistic SAT/SMT-solving
- More about using SAT/SMT

Goals Achieved?

- ✗ Attract graduate students
- ✓ Advertise automated reasoning
- ✓ Adding it to their tool-box
- Expose to new ways of thinking:
  - ✗ Encoding instead of coding
  - ✓ Calculi instead of algorithms
  - ✓ Formal and full proofs
  - ✓ Logic

Depth vs. Depth

What Does it Mean?

- More details
- Proving every detail

Example: DPLL

- Deep 1: Show full DPLL
- Deep 2: Prove everything you do

I went with Deep 2

Tools

This Year

- Z3 for SAT
- Z3 for SMT
- PySMT (with Z3) for coding

Reasons

- Will work on windows easily
- Reads cnf
- One-stop shop

Next Year

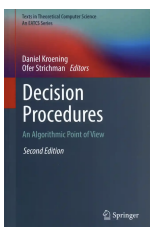
- A SAT solver for SAT
- cvc5 for SMT
- cvc5 python API for coding
- Perhaps a comparison with more tools

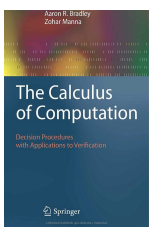
Reasons

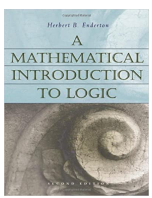
- WSL
- The right tool for the right job
- Promoting cvc5

Summary

- First-time AR class
- Depth vs. Depth
- Theory vs. Practice
- FOL in several parts (repetitions: good or bad?)
- Lessons learned



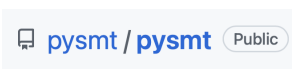




Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(*T*)

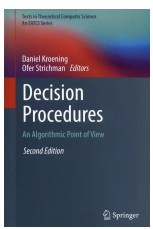
ROBERT NIEUWENHUIS AND ALBERT OLIVERAS
Technical University of Catalonia, Barcelona, Spain
AND
CESARE TINELLI
The University of Iowa, Iowa City, Iowa

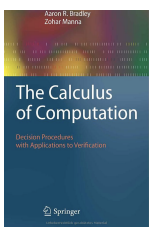


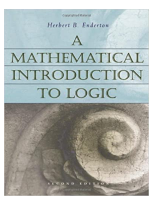


Summary

- First-time AR class
- Depth vs. Depth
- Theory vs. Practice
- FOL in several parts (repetitions: good or bad?)
- Lessons learned



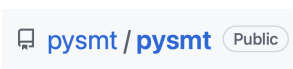




Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(*T*)

ROBERT NIEUWENHUIS AND ALBERT OLIVERAS
Technical University of Catalonia, Barcelona, Spain
AND
CESARE TINELLI
The University of Iowa, Iowa City, Iowa





Thank You