Algorithms and Satisfiability

. Mini-Project: Satisfiability Modulo Theories

Solving Problems using Satisfiability

Álvaro Torralba

AALBORG UNIVERSITET

Spring 2023

Thanks to Jörg Hoffmann for slide sources

Agenda

Agenda

Reminder: The SAT Problem

SAT

Is a propositional logic formula ϕ satisfiable?

- Propositional Logic: Satisfiability can refer to many different logics. In this course, we focus on one of the simplest!
- Satisfiable: A formula is satisfiable if it is possible to find an interpretation (assignment) that makes the formula true.

$\rightarrow$Does there exist an assignment that makes the formula true?
Examples:

- $(x \vee y) \wedge (\neg x \vee \neg y)$, Yes! $x = 0, y = 1$
- $(x \vee y) \wedge (\neg x) \wedge (\neg y)$, No!

Reminder: General Problem Solving using Satisfiability

(some new problem)

model problem in logic $\mapsto$ use off-the-shelf reasoning tool

(its solution)

- "Any problem that can be formulated as reasoning about logic."
- *Very* successful using propositional logic and modern solvers for SAT! (Propositional satisfiability testing)

$\rightarrow$**This mini-project is about how to encode problems as SAT in practice**

Our Agenda for This Chapter

How to use satisfiability tools to solve practical problems?

1. Encode problem as a logical formula
2. Use SAT solver

Our Agenda for This Chapter

How to use satisfiability tools to solve practical problems?

1. Encode problem as a logical formula
2. Use SAT solver

Apart from propositional logic, modern solvers also support more expressive formalisms:

- **Predicate Logic:** Beyond propositional logic
  $\rightarrow$Introducing quantifiers in our formulas!

Our Agenda for This Chapter

How to use satisfiability tools to solve practical problems?

1. Encode problem as a logical formula
2. Use SAT solver

Apart from propositional logic, modern solvers also support more expressive formalisms:

- **Predicate Logic:** Beyond propositional logic
  $\rightarrow$Introducing quantifiers in our formulas!
- **SAT Modulo Theories:** Extending Boolean logic with an external theory
  $\rightarrow$Increases the expressiveness of the language supported by the solver!

Our Agenda for This Chapter

How to use satisfiability tools to solve practical problems?

1. Encode problem as a logical formula
2. Use SAT solver

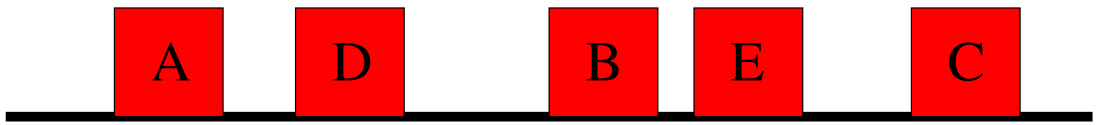
Apart from propositional logic, modern solvers also support more expressive formalisms:

- **Predicate Logic:** Beyond propositional logic
  $\rightarrow$Introducing quantifiers in our formulas!
- **SAT Modulo Theories:** Extending Boolean logic with an external theory
  $\rightarrow$Increases the expressiveness of the language supported by the solver!
- **Practical Considerations:** How to encode your problem!
  $\rightarrow$Some quick tips!

Agenda

Let's Talk About Blocks, Baby ...

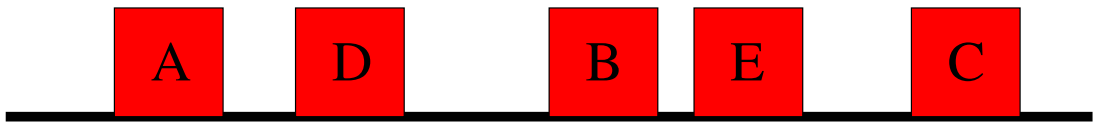
Dear students: What do you see here?



You say:

Let's Talk About Blocks, Baby ...

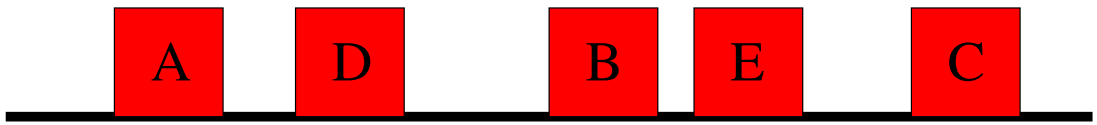
Dear students: What do you see here?



You say: "All blocks are red"; "All blocks are on the table"; "A is a block".

Let's Talk About Blocks, Baby ...

Dear students: What do you see here?



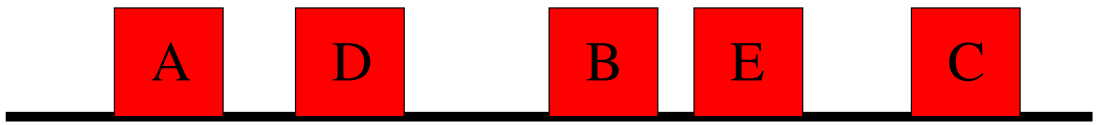
You say: "All blocks are red"; "All blocks are on the table"; "A is a block".

And now: Say it in propositional logic!

$\rightarrow$ "isRedA", "isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...

Let's Talk About Blocks, Baby ...

Dear students: What do you see here?



You say: "All blocks are red"; "All blocks are on the table"; "A is a block".

And now: Say it in propositional logic!

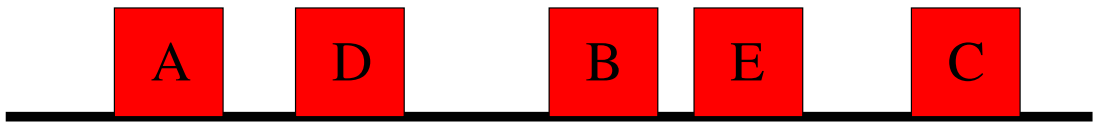
→ "isRedA", "isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...

Wait a sec! Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?

→ Could we conclude that A is red?

Let's Talk About Blocks, Baby ...

Dear students: What do you see here?



You say: "All blocks are red"; "All blocks are on the table"; "A is a block".

And now: Say it in propositional logic!

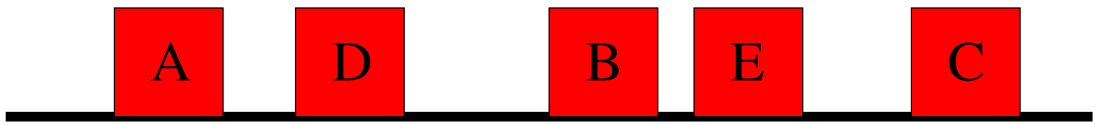
→ "isRedA", "isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...

Wait a sec! Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?

→ Could we conclude that A is red? No. These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.

Let's Talk About Blocks, Baby ...

Dear students: What do you see here?



You say: "All blocks are red"; "All blocks are on the table"; "A is a block".

And now: Say it in propositional logic!

→ "isRedA", "isRedB", ..., "onTableA", "onTableB", ..., "isBlockA", ...

Wait a sec! Why don't we just say, e.g., "AllBlocksAreRed" and "isBlockA"?

→ Could we conclude that A is red? No. These statements are atomic (just strings); their inner structure ("all blocks", "is a block") is not captured.

→ Predicate Logic extends propositional logic with the ability to explicitly speak about objects and their properties.

→ Variables ranging over objects, predicates describing object properties, ...

→ "$\forall x[Block(x) \rightarrow Red(x)]$"; "$Block(A)$"

→ We consider first-order logic, and will abbreviate PL1.

The Alphabet of PL1

Common symbols:

- Variables: x, x_1, x_2, ..., x', x'', ..., y, ..., z, ...
- Truth/Falseness: $\top$, $\bot$. (As in propositional logic)
- Operators: $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$. (As in propositional logic)
- Quantifiers: $\forall$, $\exists$.
  $\rightarrow$ Precedence: $\neg > \forall, \exists > \ldots$ (we'll be using brackets).

The Alphabet of PL1

Common symbols:

- Variables: x, x_1, x_2, ..., x', x'', ..., y, ..., z, ...
- Truth/Falseness: $\top$, $\bot$. (As in propositional logic)
- Operators: $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$. (As in propositional logic)
- Quantifiers: $\forall$, $\exists$.
  $\rightarrow$ Precedence: $\neg \ > \ \forall, \exists \ > \ \ldots$ (we'll be using brackets).

Application-specific symbols:

- Constant symbols ("object", e.g., $BlockA$, $BlockB, a, b, c, \ldots$)
- Predicate symbols, arity ≥ 1 (e.g., $Block(.)$, $Above(.,.)$)
- Function symbols, arity ≥ 1 (e.g., $WeightOf(.)$, $Succ(.)$)

Definition (Signature). *A signature Σ in predicate logic is a finite set of constant symbols, predicate symbols, and function symbols.*

Our "Silly Running Example": Lassie & Garfield

Constant symbols: $Lassie$, $Garfield$, $Snoopy$, $Lasagna$, ...

Predicate symbols: $Dog(.)$, $Cat(.)$, $Eats(.,.)$, $Chases(.,.)$, ...

Function symbols: $FoodOf(.)$, ...

Example: $\forall x[Dog(x) \rightarrow \exists y Chases(x,y)]$, which in words means

Our "Silly Running Example": Lassie & Garfield

Constant symbols: $Lassie$, $Garfield$, $Snoopy$, $Lasagna$, ...

Predicate symbols: $Dog(.)$, $Cat(.)$, $Eats(.,.)$, $Chases(.,.)$, ...

Function symbols: $FoodOf(.)$, ...

Example: $\forall x[Dog(x) \rightarrow \exists y Chases(x,y)]$, which in words means "Every dog chases something".

[We'll be showing the Lassie & Garfield example in this color and square brackets all over the place.]

Syntax of PL1

$\rightarrow$ Terms represent objects:

Definition (Term). *Let Σ be a signature. Then:*

1. *Every variable and every constant symbol is a Σ-term.* $[x,\ Garfield]$
2. *If $t_1, t_2, \ldots, t_n$ are Σ-terms and $f \in \Sigma$ is an n-ary function symbol, then $f(t_1, t_2, \ldots, t_n)$ also is a Σ-term.* $[FoodOf(x)]$

Terms without variables are ground terms. $[FoodOf(Garfield)]$

$\rightarrow$ For simplicity, we usually don't write the "Σ-".

Syntax of PL1

→ Terms represent objects:

Definition (Term). *Let Σ be a signature. Then:*

1. *Every variable and every constant symbol is a Σ-term.* $[x,\ Garfield]$
2. *If $t_1, t_2, \ldots, t_n$ are Σ-terms and $f \in \Sigma$ is an n-ary function symbol, then $f(t_1, t_2, \ldots, t_n)$ also is a Σ-term.* $[FoodOf(x)]$

Terms without variables are ground terms. $[FoodOf(Garfield)]$

→ For simplicity, we usually don't write the "Σ-".

→ Atoms represent atomic statements about objects:

Definition (Atom). *Let Σ be a signature. Then:*

1. *$\top$ and $\bot$ are Σ-atoms.*
2. *If $t_1, t_2, \ldots, t_n$ are terms and $P \in \Sigma$ is an n-ary predicate symbol, then $P(t_1, t_2, \ldots, t_n)$ is a Σ-atom.* $[Chases(Lassie, y)]$

Atoms without variables are ground atoms. $[Chases(Lassie, Garfield)]$

Syntax of PL1, ctd.

$\rightarrow$ Formulas represent complex statements about objects:

Definition (Formula). *Let Σ be a signature. Then:*

1. *Each Σ-atom is a Σ-formula.*
2. *If φ is a Σ-formula, then so is $\neg\varphi$.*

The formulas that can be constructed by rules 1. and 2. are literals.

Syntax of PL1, ctd.

$\rightarrow$ Formulas represent complex statements about objects:

Definition (Formula). *Let Σ be a signature. Then:*

1. *Each Σ-atom is a Σ-formula.*
2. *If φ is a Σ-formula, then so is $\neg\varphi$.*

The formulas that can be constructed by rules 1. and 2. are literals.

If φ and ψ are Σ-formulas, then so are:

4. $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, *and* $\varphi \leftrightarrow \psi$.

Syntax of PL1, ctd.

→ Formulas represent complex statements about objects:

Definition (Formula). *Let Σ be a signature. Then:*

1. *Each Σ-atom is a Σ-formula.*
2. *If φ is a Σ-formula, then so is $\neg\varphi$.*

The formulas that can be constructed by rules 1. and 2. are literals.

If φ and ψ are Σ-formulas, then so are:

4. $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, *and* $\varphi \leftrightarrow \psi$.

If φ is a Σ-formula and x is a variable, then

5. $\forall x\varphi$ *is a Σ-formula* ("Universal Quantification").
6. $\exists x\varphi$ *is a Σ-formula* ("Existential Quantification").

Syntax of PL1, ctd.

→ Formulas represent complex statements about objects:

Definition (Formula). *Let Σ be a signature. Then:*

1. *Each Σ-atom is a Σ-formula.*
2. *If φ is a Σ-formula, then so is $\neg\varphi$.*

The formulas that can be constructed by rules 1. and 2. are literals.

If φ and ψ are Σ-formulas, then so are:

4. $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, *and* $\varphi \leftrightarrow \psi$.

If φ is a Σ-formula and x is a variable, then

5. $\forall x\varphi$ *is a Σ-formula* ("Universal Quantification").
6. $\exists x\varphi$ *is a Σ-formula* ("Existential Quantification").

→ [E.g., $Cat(Garfield) \vee \neg Cat(Garfield)$; and $\exists x[Eats(Garfield, x)]$]

Questionnaire

Example "Animals" Σ: Constant symbols $\{Lassie, Garfield, Snoopy, Lasagna\}$; predicate symbols $\{Dog(.), Cat(.), Eats(.,.), Chases(.,.)\}$; funtion symbols $\{FoodOf(.)\}$.

Question!

Which of these are Σ-formulas?

(A): $Eats(Garfield, FoodOf(Lassie))$

(B): $Eats(Snoopy, Cat(Garfield))$

(C): $\forall x[(Dog(x) \wedge Eats(x, Lasagna)) \rightarrow \exists y(Cat(y) \wedge Chases(y, x))]$

(D): $\exists x[Dog(x) \wedge Eats(x, Lasagna) \forall y(Cat(y) \rightarrow Chases(y, x))]$

Questionnaire

Example "Animals" Σ: Constant symbols $\{Lassie, Garfield, Snoopy, Lasagna\}$; predicate symbols $\{Dog(.), Cat(.), Eats(.,.), Chases(.,.)\}$; funtion symbols $\{FoodOf(.)\}$.

Question!

Which of these are Σ-formulas?

(A): $Eats(Garfield, FoodOf(Lassie))$

(B): $Eats(Snoopy, Cat(Garfield))$

(C): $\forall x[(Dog(x) \wedge Eats(x, Lasagna)) \rightarrow \exists y(Cat(y) \wedge Chases(y, x))]$

(D): $\exists x[Dog(x) \wedge Eats(x, Lasagna) \forall y(Cat(y) \rightarrow Chases(y, x))]$

$\rightarrow$ (A), (C): Yes.

$\rightarrow$ (B): No, we can't nest predicates.

$\rightarrow$ (D): No, missing a connective between "$Eats(x, Lasagna)$" and "$\forall y(Cat(y) \rightarrow Chases(y, x))$".

The Meaning of PL1 Formulas

Example: $\forall x[Block(x) \rightarrow Red(x)],\ Block(A)$

$\rightarrow$ For all objects x, if x is a block, then x is red. A is a block.

The Meaning of PL1 Formulas

Example: $\forall x[Block(x) \rightarrow Red(x)],\ Block(A)$

$\rightarrow$ For all objects x, if x is a block, then x is red. A is a block.

More generally: (Intuition)

- Terms represent objects. $[FoodOf(Garfield) = Lasagna]$
- Predicates represent relations on the universe. $[Dog = \{Lassie, Snoopy\}]$
- Universally-quantified variables: "for all objects in the universe".
- Existentially-quantified variables: "at least one object in the universe".

PL1 as Syntactic Sugar

Whenever:

- There are no functions
- We have a finite universe Σ

We can transform a PL1 formula to an equivalent propositional formula:

1. Replace $\forall x P(x)$ by $P(a) \wedge P(b) \wedge \ldots$
2. Replace $\exists x P(x)$ by $P(a) \vee P(b) \vee \ldots$

PL1 as Syntactic Sugar

Whenever:

- There are no functions
- We have a finite universe Σ

We can transform a PL1 formula to an equivalent propositional formula:

1. Replace $\forall x P(x)$ by $P(a) \wedge P(b) \wedge \ldots$
2. Replace $\exists x P(x)$ by $P(a) \vee P(b) \vee \ldots$

Example: $\forall x[Block(x) \rightarrow Red(x)] \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Block(D) \wedge Block(E)$

PL1 as Syntactic Sugar

Whenever:

- There are no functions
- We have a finite universe Σ

We can transform a PL1 formula to an equivalent propositional formula:

1. Replace $\forall x P(x)$ by $P(a) \wedge P(b) \wedge \ldots$
2. Replace $\exists x P(x)$ by $P(a) \vee P(b) \vee \ldots$

Example: $\forall x[Block(x) \rightarrow Red(x)] \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Block(D) \wedge Block(E)$

Equivalent:

PL1 as Syntactic Sugar

Whenever:

- There are no functions
- We have a finite universe Σ

We can transform a PL1 formula to an equivalent propositional formula:

1. Replace $\forall x P(x)$ by $P(a) \wedge P(b) \wedge \ldots$
2. Replace $\exists x P(x)$ by $P(a) \vee P(b) \vee \ldots$

Example: $\forall x[Block(x) \rightarrow Red(x)] \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Block(D) \wedge Block(E)$

Equivalent: $(Block(A) \rightarrow Red(A)) \wedge (Block(B) \rightarrow Red(B)) \wedge (Block(C) \rightarrow Red(C)) \wedge (Block(D) \rightarrow Red(D)) \wedge (Block(E) \rightarrow Red(E)) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Block(D) \wedge Block(E)$

$\rightarrow$But PL1 is much more expressive in general!

Blocks, Who Cares? Let's Talk About Numbers!

Example "Integers": (A limited vocabulary to talk about these)

- The objects: $\{1, 2, 3, \dots\}$.
- Predicate 1: "$Even(x)$" should be true iff x is even.
- Predicate 2: "$Equals(x, y)$" should be true iff $x = y$.
- Function: $Succ(x)$ maps x to $x + 1$.

Blocks, Who Cares? Let's Talk About Numbers!

Example "Integers": (A limited vocabulary to talk about these)

- The objects: $\{1, 2, 3, \dots\}$.
- Predicate 1: "$Even(x)$" should be true iff x is even.
- Predicate 2: "$Equals(x, y)$" should be true iff $x = y$.
- Function: $Succ(x)$ maps x to $x + 1$.

Old problem: Say, in propositional logic, that "$1 + 1 = 2$".

$\rightarrow$ Inner structure of vocabulary is ignored (cf. "AllBlocksAreRed").

$\rightarrow$ PL1 solution: "$Equals(Succ(1), 2)$".

Blocks, Who Cares? Let's Talk About Numbers!

Example "Integers": (A limited vocabulary to talk about these)

- The objects: $\{1, 2, 3, \dots\}$.
- Predicate 1: "$Even(x)$" should be true iff x is even.
- Predicate 2: "$Equals(x, y)$" should be true iff $x = y$.
- Function: $Succ(x)$ maps x to $x + 1$.

Old problem: Say, in propositional logic, that "$1 + 1 = 2$".

$\rightarrow$ Inner structure of vocabulary is ignored (cf. "AllBlocksAreRed").

$\rightarrow$ PL1 solution: "$Equals(Succ(1), 2)$".

New problem: Say, in propositional logic, "if x is even, so is $x + 2$".

Blocks, Who Cares? Let's Talk About Numbers!

Example "Integers": (A limited vocabulary to talk about these)

- The objects: $\{1, 2, 3, \dots\}$.
- Predicate 1: "$Even(x)$" should be true iff x is even.
- Predicate 2: "$Equals(x, y)$" should be true iff $x = y$.
- Function: $Succ(x)$ maps x to $x + 1$.

Old problem: Say, in propositional logic, that "$1 + 1 = 2$".

$\rightarrow$ Inner structure of vocabulary is ignored (cf. "AllBlocksAreRed").

$\rightarrow$ PL1 solution: "$Equals(Succ(1), 2)$".

New problem: Say, in propositional logic, "if x is even, so is $x + 2$".

$\rightarrow$ It is impossible to speak about infinite sets of objects!

$\rightarrow$ PL1 solution: "$\forall x[Even(x) \rightarrow Even(Succ(Succ(x)))]$".

Now We're Talking ...

$$\forall y, x_1, x_2, x_3 \; [Equals(Plus(PowerOf(x_1, y), PowerOf(x_2, y)), PowerOf(x_3, y)) \rightarrow (Equals(y, 1) \vee Equals(y, 2))]$$

Now We're Talking ...

$$\forall y, x_1, x_2, x_3 \; [Equals(Plus(PowerOf(x_1, y), PowerOf(x_2, y)), PowerOf(x_3, y)) \rightarrow (Equals(y, 1) \vee Equals(y, 2))]$$

Theorem proving in PL1! Arbitrary theorems, in principle.

Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.

See `http://en.wikipedia.org/wiki/Automated_theorem_proving`.

Now We're Talking ...

$$\forall y, x_1, x_2, x_3 \; [Equals(Plus(PowerOf(x_1, y), PowerOf(x_2, y)), PowerOf(x_3, y)) \rightarrow (Equals(y, 1) \vee Equals(y, 2))]$$

Theorem proving in PL1! Arbitrary theorems, in principle.

Fermat's last theorem is of course infeasible, but interesting theorems can and have been proved automatically.
See `http://en.wikipedia.org/wiki/Automated_theorem_proving`.

Note: Need to axiomatize "Plus", "PowerOf", "Equals".
See `http://en.wikipedia.org/wiki/Peano_axioms`

What Are the Practical Relevance/Applications?

. . . even asking this question is a sacrilege:

What Are the Practical Relevance/Applications?

. . . even asking this question is a sacrilege: (Quotes from Wikipedia)

"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."

What Are the Practical Relevance/Applications?

. . . even asking this question is a sacrilege: (Quotes from Wikipedia)

"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."

"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."

What Are the Practical Relevance/Applications?

. . . even asking this question is a sacrilege: (Quotes from Wikipedia)

"In Europe, logic was first developed by Aristotle. Aristotelian logic became widely accepted in science and mathematics."

"The development of logic since Frege, Russell, and Wittgenstein had a profound influence on the practice of philosophy and the perceived nature of philosophical problems, and Philosophy of mathematics."

You're asking it anyhow?

- Logic programming. Prolog et al.
- Databases. Deductive databases where elements of logic allow to conclude additional facts. Logic is tied deeply with database theory.
- Semantic technology. Mega-trend since $>$ a decade. Use PL1 fragments to annotate data sets, facilitating their use and analysis.

  $\rightarrow$ Prominent PL1 fragment: Web Ontology Language OWL.

  $\rightarrow$ Prominent data set: The WWW. ($\rightarrow$ Semantic Web)

Agenda

SAT Modulo Theories

- Satisfiability Modulo Theories (SMT): extend propositional reasoning with one or more external theories

SAT Modulo Theories

- Satisfiability Modulo Theories (SMT): extend propositional reasoning with one or more external theories
- Propositional SAT: Is $(P \vee Q) \wedge (\neg P \vee \neg Q)$ satisfiable?
- Theories:
  - Equality and uninterpreted functions

$$f(x) = f(y) \wedge x \neq y$$

  - Arithmetic: rational, integer, real, ...

$$(x < y) \wedge ((y < z) \vee (x < z)) \wedge (z - 5 < x + 2)$$

  - Arrays, binary vectors, ...

Solving SMT: CDCL(T) by Example

$$(x < y) \wedge ((y < z) \vee (x < z)) \wedge (z - 5 < x + 2)$$

Solving SMT: CDCL(T) by Example

$$\underbrace{(x < y)}_{P} \wedge (\underbrace{(y < z)}_{Q} \vee \underbrace{(x < z)}_{R}) \wedge \underbrace{(z - 5 < x + 2)}_{S}$$

1. Use Boolean abstraction:

$$P \wedge (Q \vee R) \wedge S$$

Solving SMT: CDCL(T) by Example

$$\underbrace{(x < y)}_{P} \wedge (\underbrace{(y < z)}_{Q} \vee \underbrace{(x < z)}_{R}) \wedge \underbrace{(z - 5 < x + 2)}_{S}$$

1. Use Boolean abstraction:

$$P \wedge (Q \vee R) \wedge S$$

2. Use CDCL to find a (partial) satisfying assignment:

$$P \mapsto 1, Q \mapsto 1, S \mapsto 1$$

Solving SMT: CDCL(T) by Example

$$\underbrace{(x < y)}_{P} \wedge (\underbrace{(y < z)}_{Q} \vee \underbrace{(x < z)}_{R}) \wedge \underbrace{(z - 5 < x + 2)}_{S}$$

1. Use Boolean abstraction:

$$P \wedge (Q \vee R) \wedge S$$

2. Use CDCL to find a (partial) satisfying assignment:

$$P \mapsto 1, Q \mapsto 1, S \mapsto 1$$

3. Call theory solver to check if it satisfies the external theory: If yes, we are done. Otherwise, learn a new clause: $(\neg P \vee \neg Q \vee \neg S)$

   Remark: This clause learning should learn a clause as informative as possible by using a justification procedure as in Chapter 8.

Solving SMT: CDCL(T) by Example

$$\underbrace{(x < y)}_{P} \wedge (\underbrace{(y < z)}_{Q} \vee \underbrace{(x < z)}_{R}) \wedge \underbrace{(z - 5 < x + 2)}_{S}$$

1. Use Boolean abstraction:

$$P \wedge (Q \vee R) \wedge S$$

2. Use CDCL to find a (partial) satisfying assignment:

$$P \mapsto 1, Q \mapsto 1, S \mapsto 1$$

3. Call theory solver to check if it satisfies the external theory: If yes, we are done. Otherwise, learn a new clause: $(\neg P \vee \neg Q \vee \neg S)$

   Remark: This clause learning should learn a clause as informative as possible by using a justification procedure as in Chapter 8.

4. Go to 2

Useful Resources

SMT Competition: `https://smt-comp.github.io`

SMT-Lib: `https://smtlib.cs.uiowa.edu/`

There are many SMT solvers available. To name a few:

- Z3: `https://github.com/Z3Prover/z3`
- CVC4: `https://cvc4.github.io/`
- Yices 2: `https://github.com/SRI-CSL/yices2`

Format SMT-LIB2

- Choose the theory to be used:

```
(set_logic QF_LIA)
```

- Some theories:
  - QF_ LRA: linear real arithmetic without quantifiers
  - QF_ LIA: linear integer arithmetic without quantifiers
- SMT-LIB 2 does not allow us to mix problems, although some of the solvers allow it

Format SMT-LIB2

Declare variables:

```
(declare-fun x () Int)
(declare-fun z 1 3 () Real)
```

Add formulas: expressions are written in prefix form:
$(< operator >< arg1 > \cdots < argn >)$

```
(assert (or (<= (+ x 3) (* 2 u))
            (>= (+ v 4) y)
            (>= (+ x y z ) 2)))
```

Format SMT-LIB2

- Ask the solver to verify satisfiability

```
(check-sat)
```

- And report the model

```
(get-model)
```

Agenda

How to Prove Something

Typical Exercise: Given X, Y, and Z, prove φ to be true

- Knowledge Base KB: a formula KB that we know to be true $(X \wedge Y \wedge Z)$
- Statement that we want to prove: φ

How to Prove Something

Typical Exercise: Given X, Y, and Z, prove φ to be true

- Knowledge Base KB: a formula KB that we know to be true $(X \wedge Y \wedge Z)$
- Statement that we want to prove: φ

Proof by contradiction: Assume $\neg\varphi$ and show that this leads to a contradiction

How to Prove Something

Typical Exercise: Given X, Y, and Z, prove φ to be true

- Knowledge Base KB: a formula KB that we know to be true $(X \wedge Y \wedge Z)$
- Statement that we want to prove: φ

Proof by contradiction: Assume $\neg\varphi$ and show that this leads to a contradiction

$\rightarrow$ Deduction can be reduced to proving unsatisfiability of KB $\wedge\neg\varphi$: "Assume, to the contrary, that KB holds but φ does not hold; then derive False".

How to Prove Something

Typical Exercise: Given X, Y, and Z, prove φ to be true

- Knowledge Base KB: a formula KB that we know to be true $(X \wedge Y \wedge Z)$
- Statement that we want to prove: φ

Proof by contradiction: Assume $\neg\varphi$ and show that this leads to a contradiction

$\rightarrow$ Deduction can be reduced to proving unsatisfiability of KB $\wedge\neg\varphi$: "Assume, to the contrary, that KB holds but φ does not hold; then derive False".

$\rightarrow$You also need to check that KB is satisfiable, otherwise, if there is a contradiction in KB you can prove any statement!

Sometimes Less is More

There are a whole lot of theories, which one should I use?

Sometimes Less is More

There are a whole lot of theories, which one should I use?

Trade-off between Expresiveness and Complexity:

- SAT is NP
- PL1 is undecidable
- SMT depends on the theory

Sometimes Less is More

There are a whole lot of theories, which one should I use?

Trade-off between Expresiveness and Complexity:

- SAT is NP
- PL1 is undecidable
- SMT depends on the theory

→Always try to express your problem in a way that's as simple as possible! (e.g. propositional SAT without any theories)

The solvers available to reason about arithmetical constraints are wildly different depending on what fragments of arithmetic is used. We summarize the main fragments, available decision procedures, and examples in Table 1 where x, y range over reals and a, b range over integers.

Logic	Description	Solver	Example
LRA	Linear Real Arithmetic	Dual Simplex [28]	$x + \frac{1}{2}y \leq 3$
LIA	Linear Integer Arithmetic	Cuts + Branch	$a + 3b \leq 3$
LIRA	Mixed Real/Integer	[7, 12, 14, 26, 28]	$x + a \geq 4$
IDL	Integer Difference Logic	Floyd-Warshall	$a - b \leq 4$
RDL	Real Difference Logic	Bellman-Ford	$x - y \leq 4$
UTVPI	Unit two-variable / inequality	Bellman-Ford	$x + y \leq 4$
NRA	Polynomial Real Arithmetic	Model based CAD [42]	$x^2 + y^2 < 1$
NIA	Non-linear Integer Arithmetic	CAD + Branch [41]	$a^2 = 2$
		Linearization [15]	

Solving Numerical Problems

Linear Programming

- Allows you to handle numerical problems of the form:

$$\begin{aligned} \text{minimize} \quad & x + 5y \\ \text{subject to} \quad & x \leq 3 \\ & 2x + y \leq 4 \end{aligned}$$

Solving Numerical Problems

Linear Programming

- Allows you to handle numerical problems of the form:

$$\begin{aligned} \text{minimize} \quad & x + 5y \\ \text{subject to} \quad & x \leq 3 \\ & 2x + y \leq 4 \end{aligned}$$

Mixed Integer Linear Programming:

- Allows to define binary (Boolean) and Integer variables
- Can also encode SAT

$$p \vee q \text{ encoded as } x_p + x_q \geq 1$$

Solving Numerical Problems

Linear Programming

- Allows you to handle numerical problems of the form:

$$\begin{aligned} \text{minimize} \quad & x + 5y \\ \text{subject to} \quad & x \leq 3 \\ & 2x + y \leq 4 \end{aligned}$$

Mixed Integer Linear Programming:

- Allows to define binary (Boolean) and Integer variables
- Can also encode SAT

$$p \vee q \text{ encoded as } x_p + x_q \geq 1$$

$\rightarrow$ If your problem is mostly numerical, then LP or MILP are good alternatives too!