Decision Procedures

An Algorithmic Point of View

Basic Concepts and Background

D. Kroening O. Strichman

ETH/Technion

Version 1.1, 2007

Part I

Basic Concepts and Background

Outline

Logic in Computer Science

- Reasoning in AI
- Proofs in verification
- Queries in Databases
- . . . many more

What is Logic?

Some useful definitions from the web:

- "*Science dealing with the principles of valid reasoning and argument*"
- "*A formal and powerful method of explaining why the program doesn't work*"
- "*The art of being wrong with confidence*"

So what is Logic?

- Defined by
  - Syntax
    (including the Signature Σ of the logic: variables and their domain, function and predicate symbols, quantifiers, etc.)
  - Semantics: Axioms and Inference rules
- A logic allows us *to infer theorems*

Example: Propositional Logic

- Syntax

  formula : *Boolean-var* | $\neg$*formula* | *formula* $\vee$ *formula* | (*formula*) | T | F

  (Syntactic sugar: *formula* $\wedge$ *formula* | *formula* $\longrightarrow$ *formula* ...)

- Axioms:
  1. $\vdash (A \longrightarrow (B \longrightarrow A))$
  2. $\vdash ((A \longrightarrow (B \longrightarrow C)) \longrightarrow ((A \longrightarrow B) \longrightarrow (A \longrightarrow C)))$
  3. $\vdash (\neg B \longrightarrow \neg A) \longrightarrow (A \longrightarrow B)$
- (Single) Inference Rule: Modus Ponens (MP)

$$\frac{\vdash A \qquad \vdash A \longrightarrow B}{\vdash B}$$

A specific (one of many possible) *Deductive System* for Propositional Logic. It is known as the Hilbert System $\mathcal{H}$.

A proof by deduction: example

- Notation: $\vdash_{\mathcal{H}} \varphi$ 'there exists a proof of φ in $\mathcal{H}$'
- Theorem: $\vdash_{\mathcal{H}} (A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$
- Proof:

1.	$\{A \longrightarrow B, B \longrightarrow C, A\} \vdash_{\mathcal{H}} A$	Premise
2.	$\{A \longrightarrow B, B \longrightarrow C, A\} \vdash_{\mathcal{H}} A \longrightarrow B$	Premise
3.	$\{A \longrightarrow B, B \longrightarrow C, A\} \vdash_{\mathcal{H}} B$	M.P. 1,2
4.	$\{A \longrightarrow B, B \longrightarrow C, A\} \vdash_{\mathcal{H}} B \longrightarrow C$	Premise
5.	$\{A \longrightarrow B, B \longrightarrow C, A\} \vdash_{\mathcal{H}} C$	M.P. 3,4
6.	$\{A \longrightarrow B, B \longrightarrow C\} \vdash_{\mathcal{H}} (A \longrightarrow C)$	Deduction 5
7.	$\{A \longrightarrow B\} \vdash_{\mathcal{H}} ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$	Deduction 6
8.	$\vdash_{\mathcal{H}} (A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$	Deduction 7

More on Semantics

Can be given via

- axioms and inference rules, or
- using another (background) logic, or
- via truth tables:

x_1	x_2	$x_1 \wedge x_2$	$x_1 \vee x_2$	...
T	T	T	T	
T	F	F	T	
F	T	F	T	
F	F	F	F	

Satisfying Interpretations

- If an assignment α satisfies (according to the truth tables) a formula φ, we write: $\alpha \models \phi$.
- Example:

$$\varphi : \neg(x_1 \wedge \neg(x_2 \vee \neg x_3))$$

- Assignments for the example:
  - $\alpha_1 : (x_1 = \mathsf{T}, x_2 = \mathsf{F}, x_3 = \mathsf{F})$
  - $\alpha_2 : (x_1 = \mathsf{T}, x_2 = \mathsf{F}, x_3 = \mathsf{T})$
  - $\alpha_1 \models \varphi$, but $\alpha_2 \not\models \varphi$

Satisfiability, Validity, etc.

Definition (Satisfiable)

A formula φ is *satisfiable* if $\exists \alpha.\, \alpha \models \varphi$.

Definition (Valid)

A formula φ is *valid* if $\forall \alpha.\, \alpha \models \varphi$. If φ is valid, we write $\models \varphi$.

Observation: φ is valid if and only if $\neg\varphi$ is unsatisfiable.

A proof by enumeration: same example

A	B	C	$(A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

$$\models (A \longrightarrow B) \longrightarrow ((B \longrightarrow C) \longrightarrow (A \longrightarrow C))$$

Soundness and completeness of a deductive system

Given a deductive system $\mathcal{D}$,

- $\mathcal{D}$ is sound for a logic $\mathcal{L}$, if for every formula f in $\mathcal{L}$,

$$\vdash_{\mathcal{D}} f \longrightarrow \models f$$

I.e., all formulas proven by the deductive system are valid.

- $\mathcal{D}$ is complete if for every formula f in $\mathcal{L}$,

$$\models f \longrightarrow \vdash_{\mathcal{D}} f$$

I.e., the deductive system can prove all valid formulas.

The decision problem

Definition (Decision Problem)

The decision problem for a formula: given φ, is φ valid?

Definition (Decision Procedure)

A *decision procedure* for a logic is an algorithm that solves the decision problem for any formula in this logic.

We are naturally interested in a sound and complete decision procedure.

Soundness and Completeness

What does it mean that a decision procedure is sound and complete?

- Soundness: the answer returned by the decision procedure is always correct
  (Question: "correct" according to what?)

- Completeness: returns with a yes/no answer in finite time.
  (Question: How does this definition relate to the definition of completeness of a deduction system?)

Soundness and Completeness

- Soundness: "when I say that it rains, it rains, and when I say it doesn't rain, it doesn't rain"
- Completeness: "When asked, I always reply (in a finite time) whether it rains"

Definition

A logic is decidable $\Longleftrightarrow$
there is a sound and complete algorithm that decides if a well-formed expression in this logic is valid.

Soundness and Completeness (cont'd)

Algorithm #1: for checking if it rains outside:
stand right outside the door and say 'it rains'

- It is *not sound* because you might say it rains when it doesn't.
- But it is complete: you *always get an answer* in a finite time.

Soundness and Completeness (cont'd)

Algorithm #2 for checking if it rains outside:
stand right outside the door and say 'it rains'
if and only if you feel the rain.

- It is *sound* because you say it rains only if it actually rains.
- It is incomplete because you do not say anything if it doesn't rain (we do not know whether it doesn't rain, or it takes the person too long to answer ...).

Decidability

- Propositional logic is *decidable*
  $\Rightarrow$ there is a sound and complete algorithm (e.g., truth tables) to decide whether a propositional formula is valid.
- Arithmetic over integers is . . .?. . . *undecidable*
  (this is Gödel's incompleteness result)

Inference engines

- We saw that in Propositional Logic we can infer using either a deductive system ('deduction') or truth tables ('enumeration').
- Which, in the general case, is the better method?
- All logics have a deductive definition.
- NOT all logics can be decided with an enumerative method.

Deduction vs. enumeration

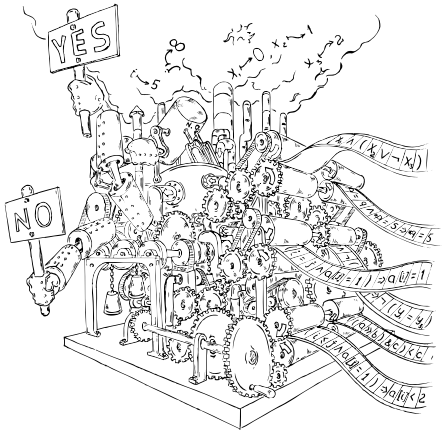
Deductive methods
Axioms and inference rules

Requires thinking...

Enumerative methods
Truth tables

Requires pressing 'Enter'...

Whenever we can: build an engine to think for us



Expressiveness of a logic

- Each formula defines a language:
  the set of satisfying assignments ('models') are the words accepted by this language.
- Consider the logic '2-CNF'

  formula : *literal* $\vee$ *literal* | *formula* $\wedge$ *formula*
  literal : *Boolean-variable* | $\neg$*Boolean-variable*
- A '2-CNF' formula: $(x_1 \vee \neg x_2) \wedge (\neg x_3 \vee x_2)$

Expressiveness of a logic

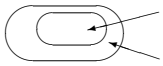
- Now consider the Propositional Logic formula

$$\varphi : x_1 \vee x_2 \vee x_3$$

- Q: Can we express this language with 2-CNF?
- A: No.

Proof.
The language accepted by φ has 7 words: all assignments other than $x_1 = x_2 = x_3 = \mathrm{F}$. The first 2-CNF clause removes $\frac{1}{4}$ of the assignments, which leaves us with 6 accepted words. Additional clauses only remove more assignments. □

Expressiveness of a logic



$\mathcal{L}_2$ is more expressive than $\mathcal{L}_1$.
Notation: $\mathcal{L}_1 \prec \mathcal{L}_2$

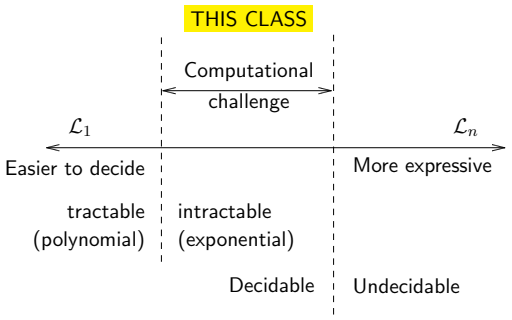
Claim

2-CNF $\prec$ Propositional Logic

Generally, there is only a partial order between logics.

Tradeoff: expressiveness/computational hardness

Assume we are given logics $\mathcal{L}_1 \prec \ldots \prec \mathcal{L}_n$



Expressiveness and complexity

- Q1: Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two logics whose satisfiability problem is decidable and in the same complexity class.
  Is the satisfiability problem of an $\mathcal{L}_1$ formula reducible to a satisfiability problem of an $\mathcal{L}_2$ formula?
- Q2: Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two logics whose satisfiability problems are reducible to one another.

  Are $\mathcal{L}_1$ and $\mathcal{L}_2$ in the same complexity class?

When is a specific logic useful?

1. Expressible enough to state something interesting.
2. Decidable (or semi-decidable) and more efficiently solvable than richer logics.
3. More expressible, or more natural for expressing some models in comparison to 'leaner' logics.

Example: First Order Peano Arithmetic

- Constants: $0, 1$
- Function symbols: '$+$', '$*$', predicate symbol: '$=$'
- Domain: natural numbers
- Semantics given by axioms:
  1. $\forall x : (0 \neq x + 1)$
  2. $\forall x, y : (x \neq y) \longrightarrow (x + 1 \neq y + 1)$
  3. Induction
  4. $\forall x : x + 0 = x$
  5. $\forall x, y : (x + y) + 1 = x + (y + 1)$
  6. $\forall x : x * 0 = 0$
  7. $\forall x, y : x * (y + 1) = x * y + x$

UNDECIDABLE!

Example: Presburger Arithmetic

- Constants: $0, 1$
- Function symbols: '$+$', predicate symbol: '$=$'
- Domain: natural numbers
- Semantics given by axioms:
  1. $\forall x : (0 \neq x + 1)$
  2. $\forall x, y : (x \neq y) \longrightarrow (x + 1 \neq y + 1)$
  3. Induction
  4. $\forall x : x + 0 = x$
  5. $\forall x, y : (x + y) + 1 = x + (y + 1)$
  6. ~~$\forall x : x * 0 = 0$~~
  7. ~~$\forall x, y : x * (y + 1) = x * y + x$~~

DECIDABLE!

Some notes on Propositional Logic

- The simplest of them all
- NP-complete
- Exceptionally efficient solvers (SAT engines, BDDs)
- Formulas with 10^5 variables are being solved regularly
- All the logics that we will consider can be reduced directly to this logic

Some notes on Propositional Logic

v positive literal $\quad \neg v$ negative literal

- Also known as 'the phase', or 'the polarity' of the literal.
- The 'logical phase' of a literal can be computed by counting the number of negations that nest it.
  - v is logically negative in: $\neg v$, $\neg(\neg(\neg v))$, $v \longrightarrow u$, $\neg(u \longrightarrow v)$
  - v is logically positive in: v, $\neg(v \longrightarrow u)$

Normal forms

- Conjunctive Normal Form (CNF)

$$\bigwedge \bigvee l_i$$

- Disjunctive Normal Form (DNF)

$$\bigvee \bigwedge l_i$$

  Satisfiability is in P!
- Negation Normal Form (NNF)
  (all negations are over literals, not sub-formulas)

Note that CNF and DNF are special cases of NNF.

Normal forms

Conversion into normal forms:

- Convert φ to a CNF: with additional variables, in polynomial time
- Convert φ to DNF: exponential time and space
- Convert φ to NNF: polynomial time

The 'Pure literal rule'

- Consider $\varphi : (x \vee y) \wedge (\neg x \vee z) \wedge (x \vee y \vee \neg z)$
- y is 'pure': it only appears in one phase
- Idea: when trying to satisfy φ, first assign $y =$ true

- Why? If there is a satisfying assignment to φ, there is a satisfying assignment in which $y =$ true.
- Generalization: assign all pure literals according to their phase

Pure literals in NNF

- CNF is a special case of NNF
- A pure literal in NNF is defined in the same way:
  a literal that only appears in one phase.
- We can always start satisfiability checking by assigning these pure literals true or false according to their phase.
- We will rely on a similar principle also when considering other logics.

Monotonicity of NNF

Theorem

NNF formulas are monotonically satisfied.

In case of CNF, this is simply the pure literal rule.

Example

$$\begin{array}{lccccccc} \varphi: & (x_1 & \wedge & \neg x_2) & \vee & (x_2 & \vee & (x_3 \wedge x_1)) \\ \alpha: & 0 & & 1 & & 1 & 1 & 0 \\ \alpha': & 1 & & & & & & 1 \end{array}$$

Monotonicity of NNF

Why is monotonicity relevant to decision procedures?

- We will use the fact that if we make unsatisfied predicates satisfied, we do not make the formula unsatisfied.
- We will rely heavily on this fact later: it simplifies decision procedures.