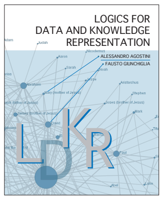
Logics for Data and Knowledge Representation

Alessandro Agostini
agostini@dit.unitn.it

Fausto Giunchiglia
fausto@dit.unitn.it

University of Trento

Outline



- Introduction
- Syntax
- Semantics
  - intensions
  - entailment
- Reasoning
  - DP Procedure

Introduction

- Despite all the several representation languages, all *knowledge* representation languages deal with sentences.
- Sentences are propositions (by definition).
- Propositional logic deals with propositions.
- As propositional logic is the simplest logic that does this, it is the right place to start.

Language

Language (Syntax)

- The first step in setting up a formal language (viz. a propositional language, PL) is to list the symbols, that is, the alphabet of symbols.
- We denote the alphabet of a propositional language by Σ_0. So, PL is a symbolic language.
- Symbols in Σ_0 are divided in 'descriptive' (nonlogical) and 'non-descriptive' (logical).

Logical Symbols (non-descriptive)

- 1. Propositional constants: A, B, ...
- 2. Logical constants ("Boolean operators"): $\wedge, \vee, \neg$
- 3. Parentheses (auxiliary symbols): (,)
- 4. Propositional variables: P, Q, ..., ψ, θ...
- Remark: not only characters but also words like e.g. "monkey", "LDKR," etc. are in 4.

Defined Symbols

- Logical *defined* constants are, for all P:
  - ⊥ (falsehood symbol): $\bot =_{df} P\wedge\neg P$
  - ⊤ (truth symbol): $\top =_{df} \neg\bot$
- **Remark**:
  Defined symbols are *not* strictly necessary.
- They increase the usability by reducing the syntactic complexity of propositions.

Formation Rules (FR)

- Atomic Formulas (atomic propositions):
  1. A, B,..., P, Q,...; ⊥, ⊤.
- Propositional Formulas (propositions), also called well-formed formulas, in short wff's):
  2. All the atomic formulas.
  3. ¬P, P∧Q, P∨Q for all wff's P, Q.
- Σ_0 + FR define a propositional language.

Remark

- Propositions (wff's in PL) are the "correct" formal expression to be built from Σ_0.
- **Example:**
  - 'inLab(Monkey)' or also 'inLab-Monkey', 'inlab-monkey', 'in-lab-monkey' are correct (with meaning T in our example);
  - 'inLab(high)', 'inLab(Monkey)∧', 'P ¬∧Q'

  are not correct (i.e. with no meaning).

Important Remark!

- Observe the two equivalent propositions:
  - 'inLab(Monkey)' (*) and 'inLab-Monkey'.
  - 'Monkey' does not represent an individual of some domain; we have not such an expressiveness in PL! (We need class-PL.)
  - Parentheses in 'inLab(Monkey)' are not the auxiliary symbols like in e.g. (P∧¬Q) ∨ Q
  - The semantics of (*) in PL and FOL differ!

Example (MB)

- Take the Monkey-Bananas problem.
- Some significant data are: the monkey, the bananas, the box, positions A, B...
- Significant knowledge is: Low, High, Go, ClimbOn...
- All other data and knowledge are irrelevant!

"There is a monkey in a laboratory with some bananas hanging out of reach from the ceiling. A box is available that will enable the monkey to reach the bananas if he climbs on it. Initially, the monkey is at A, the bananas at B, and the box at C. The monkey and box have height Low, but if the monkey climbs onto the box he will have height High, the same as the bananas. [...]"

Example (MB, cont')

- Language: 'Monkey', 'Bananas', 'Box', 'Ceiling', 'A', 'B', 'C'; 'inLab-Monkey', 'inLab-Bananas', 'inLab-Box', 'high-Bananas', 'low-Monkey', ...
  - blue wff's: data, red wff's: knowledge.
- **Important Remark**:
  In PL we cannot reason by using 'Monkey' and 'inLab-Monkey' to conclude something on the monkey as an individual in the world.

Example (MB, cont')

- Intended interpretation (informal semantics):
  - 'Monkey' is intended to be a monkey,
  - 'Bananas' is intended to be some bananas
  - ...
  - 'inLab-Monkey' is intended to say that a monkey is in a laboratory...
  - ...
  - 'high-Bananas' is intended to say that some bananas are high to be reached, ... etc.

data

knowledge

Propositional Theory

- Definition. A set of propositions is a propositional (or sentential) theory.
- **NB:** a propositional theory is a (propositional) knowledge base.
- Recall that:
  - knowledge is a "statement"
  - we assumed statements are propositions.

Propositional Theories and Databases

- A propositional theory is not a data base!
  - Propositions don't show data explicitely
    - Example: In 'inLab(Monkey)', 'Monkey' does not refer to an individual from some data domain.
- In PL the form P(*a*) must be interpreted as P-*a* (so the use of form P(*a*) is ambiguous).

Example + Remark

- **Example:** A propositional theory, i.e., a knowledge base (KB) for the MB problem is a set like this:

  {inLab-Monkey, inLab-Bananas, inLab-Box, high-Bananas, low-Monkey, isAtMonkey-A, isAtBananas-B, isAtBox-C, ClimbMonkeyBox∧isAtBox-B)→reachMonkey-Bananas,...}.

- **Remark:** The notion of theory / KB is a syntactic notion (i.e. no meaning in symbols).

Semantics

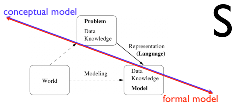
Semantics

- So far the elements of our propositional language are simply strings of symbols

  without formal meaning

- The meanings which are intended to be attached to the symbols and propositions form the intended interpretation of the language (viz. its symbols, formulas, etc.).

Semantics



- We must make sure to assign the formal meanings out of our intended interpretation to the language, so that formulas (propositions) express what we intended.
- This is done by defining a formal model M.
- Technically: we have to define a pair (M, |=) for our propositional language (see below).

Intensional Semantics Intensions (Definition)

- Definition. The intension of a proposition is the sum of all the properties that *must* be possessed by every particular to which the proposition can be applied.

The intension consists of all the *properties* the proposition *implies* (Rescher, 1964).

Example

- Take the proposition 'airplane':
- Airplanes may differ in many ways, but anything to which one properly refers to by using the proposition 'airplane' will have to have certain specifiable properties, e.g. be self-propelled, man-made, having wings, etc.
- These specifiable properties go to define the intension of the proposition 'airplane'.

Intensional Interpretation

- Definition. The intensional interprertation I of a proposition P, denoted by I(P), is the intension of P.
- **Example 1**: I('airplane') = = {self-propelled, man-made, having wings, ...}
- **Example 2**: I('inLab-Monkey') = = {there is a monkey, there is a lab, the monkey is in the lab}

Intensions versus Truth-Values

- The intensional interpretation of a proposition determines its truth-value.
- **Example**: Proposition 'airplane' is true iff airplanes are
  - self-propelled and
  - man-made and
  - have wings or
  - fly and
  - are not birds, ...

Truth-values

- In intensional semantics, the central notion is that of 'truth valuation'.
- Definition. A truth valuation on a propositional language L is a mapping ν assigning to each formula P of L a truth value ν(P), i.e., a member of the set {True, False} (in short {T, F}), defined as follows.

[see the next slide]

Truth-valuation ν

- ν(¬P) = True iff ν(P)=False
- ν(P∧Q) = True iff ν(P)=True and ν(Q)=True
- ν(P∨Q) = True iff ν(P)=True or ν(Q)=T
- ν(⊥) = False (since $\bot =_{df} P\wedge\neg P$)
- ν(⊤) = True (since $\top =_{df} \neg\bot$)

Truth Tables

- To compute (in polynomial time) truth valuations, the method of Truth Tables was introduced (Wittgenstein, 1921).
- Truth tables are well-know:

P	Q	$\neg P$	$P\wedge Q$	$P\vee Q$
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Truth-table (TT) for the logical constants ¬ , ∧, ∨

Truth Relation (Satisfaction Relation)

- Let ν be a truth valuation on language L.
  - We define the truth-relation (or satisfaction-relation) |= and write
    ν |= P
    (read: ν satisfies P) iff ν(P) = True.
  - Given a set of propositions Γ, we define
    ν |= Γ
    iff if ν |= θ for all θ ∈ Γ.

Model, Satisfiable

- Definition. Let ν be a truth valuation on language L.
  - ν is a model of a proposition P (set of propositions Γ) iff ν satisfies P (Γ).
- P (Γ) is satisfiable if there is some truth valuation ν such that ν |= P (ν |= Γ).

Truth and Validity

- Definition. Let ν be a truth valuation:
  (1) P is true under ν if ν |= P.
  (2) P is valid (and P is called a tautology), if ν |= P for all ν (notation: |= P).
- Note that the notions of 'true' and 'false' are relative to some truth valuation (cf. 'under').
- **Fact**: A proposition is true iff is satisfiable.

Truth Relation and Reasoning Services

- The basic reasoning tasks (or "services") we can represent (and compute) using |= are:
- **Model Checking (EVAL)**: Is a proposition P true under a truth-valuation ν (i.e., is ν a model of P: ν |= P)?
- **Example**: P = 'inLab-monkey∨inLab-box'

  P is true under any truth-valuation ν that assigns True to '¬LabEmpty'.

Truth Relation and Reasoning Services

- **Validity**: Is a proposition P true under *every possible* truth-valuation ν?
- **Example**: P = 'Being∨¬Being'.

  P is valid, since for every truth-valuation ν,
  either
  ν('Being') = T, so ν(P) = T;
  or
  ν('Being') = F, so ν(¬'Being') = T, i.e. ν(P) = T.

Truth Relation and Reasoning Services

- **Satisfiability (SAT)**: Is P satisfiable?
- **Example**: 1. 'Being∨¬Being' is satisfiable.

  2. P = 'A∧¬A' is unsatisfiable (inconsistent):

  for every truth-valuation ν,
  either
  ν(A) = T, so ν(¬A) = F, hence ν(P) = F;
  or
  ν(A) = F, so ν(¬A) = T, hence ν(P) = T.

Propositional Entailment

- Definition. A set Γ (eventually empty) of propositions entails a proposition ψ (written: Γ |= ψ) if for all ν,
  if ν |= θ for all θ ∈ Γ, then ν |= ψ.
- If Γ |= ψ, then we say that ψ is a logical consequence of Γ, and Γ logically implies ψ.
- Remark: The entailment is a relationship between wff's that is based on semantics

Examples

- Suppose P = 'today'∨'tomorrow'. Then:
  - {} |= P (notation: |= P)
  - {'today'} |= 'today'∨'tomorrow'
  - {'today'} not-|= 'tomorrow'
- Exercise:
  (a) Define Γ and P such that Γ |= P.
  (b) Define Γ and P such that Γ not-|= P.

Reductio ad Absurdum

- In the finite, we can formally relate the propositional entailment to satisfiability.
- Theorem: $\{\theta_1, ..., \theta_n\} \models \psi$ if and only if $\{\theta_1, \theta_2, ..., \theta_n, \neg\psi\}$ is unsatisfiable.
  - Proof: exercise.
- The theorem fixes the way to prove ψ by *reductio ad absurdum*.

Implication Premise

- Propositional entailment in the finite, i.e.:

  $\{\theta_1, ..., \theta_n\} \models \psi$

  ($\{\theta_1, ..., \theta_n\}$ finite set of propositions) can be viewed as the logical implication

  $(\theta_1 \wedge \theta_2 \wedge ... \wedge \theta_n) \rightarrow \psi$

  [$(\theta_1 \wedge \theta_2 \wedge ... \wedge \theta_n)$ logically implies ψ]

Implication (Syntax)

- We extend our alphabet of symbols with the *defined* logical constants:
  → (implication)
  ↔ (double implication or equivalence)
- **Remark**: ⊥ and ⊤ are syntactically defined symbols: $\bot =_{df} P \wedge \neg P$, $\top =_{df} \neg\bot$,
  →,↔ are semantically defined symbols!

Implication (Semantics)

- P→Q, P↔Q are wff's for all wff's P, Q.
- Let propositions ψ, θ, and <u>*finite*</u> set $\{\theta_1,...,\theta_n\}$ of propositions be given. We define:
  - $\models \theta \to \psi$ ($\theta \to \psi$ <u>valid</u>) iff $\theta \models \psi$
  - $\models (\theta_1 \wedge ... \wedge \theta_n) \to \psi$ iff $\{\theta_1,...,\theta_n\} \models \psi$
  - $\models \theta \leftrightarrow \psi$ iff $\theta \to \psi$ and $\psi \to \theta$.

Truth Table of →

- By the definition of the semantics of → in terms of |= we have the following truth-table for the logical implication P→Q:

P	Q	$P \models Q$	$P \to Q$
False	False	True	True
False	True	True	True
True	False	False	False
True	True	True	True

Logical Implication Three Properties

- **P1**: For all ψ,θ,
  $\psi \to \theta$ iff $\neg\psi \vee \theta$.
- **P2**: For all ψ,
  $\neg\psi$ iff $\psi \to \bot$.
- **P3**: For all ψ,
  $\bot \models \psi$
  (inconsistent theories imply <u>any</u> proposition)

Examples

- Laws for ∧, ¬ and →:
  - $(A \wedge \neg B) \to \neg(A \to B)$
  - $\neg(A \to B) \to (A \wedge \neg B)$
- Pierce's law: $((A \to B) \to A) \to A$
- De Morgan's laws:
  1. $\neg(A \vee B) \to (\neg A \wedge \neg B)$; $(\neg A \wedge \neg B) \to \neg(A \vee B)$
  2. $\neg(A \wedge B) \to (\neg A \vee \neg B)$; $(\neg A \vee \neg B) \to \neg(A \wedge B)$

Exercise

- Prove the following double implications (i.e., <u>logical equivalences</u>) by using truth tables.
  - $(\alpha \wedge \beta) \leftrightarrow (\beta \wedge \alpha)$ commutativity of ∧
  - $(\alpha \vee \beta) \leftrightarrow (\beta \vee \alpha)$ commutativity of ∨
  - $((\alpha \wedge \beta) \wedge \gamma) \leftrightarrow (\alpha \wedge (\beta \wedge \gamma))$ associativity of ∧

Exercise (cont')

- $((\alpha \vee \beta) \vee \gamma) \leftrightarrow (\alpha \vee (\beta \vee \gamma))$
  associativity of $\vee$
- $\neg(\neg\alpha) \leftrightarrow \alpha$ double-negation elimination
- $(\alpha \rightarrow \beta) \leftrightarrow (\neg\beta \rightarrow \neg\alpha)$ contraposition
- $(\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \vee \beta)$ $\rightarrow$-elimination
- $(\alpha \leftrightarrow \beta) \leftrightarrow ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$ $\leftrightarrow$-elimination

Exercise (cont')

- $\neg(\alpha \wedge \beta) \leftrightarrow (\neg\alpha \vee \neg\beta)$ DeMorgan Law for $\wedge$
- $\neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \neg\beta)$ DeMorgan Law for $\vee$
- $(\alpha \wedge (\beta \vee \gamma)) \leftrightarrow ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$
  distributivity of $\wedge$ over $\vee$
- $(\alpha \vee (\beta \wedge \gamma)) \leftrightarrow ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$
  distributivity of $\vee$ over $\wedge$

Reasoning

Reasoning Services

- Basic reasoning tasks for a PL-based system :
  - **Entailment**: Is a proposition ψ true if the world represented by a propositional theory KB is true? I.e.: $KB \models \psi$?
  - **Consistency**: Is ψ consistent in a KB? I.e.: $KB \cup \{\psi\} \models \bot$?
  - **Satisfiability (SAT)**: Is KB satisfiable?

PSAT-Problem (Boolean SAT)

- Definition.
  PSAT = $\{P \in PL \mid \text{exists } \nu \text{ such that } \nu \models P\}$.
- Satisfiability Problem: Is PSAT decidable?
- Theorem [Cook,1971] PSAT is NP-complete

  The theorem established a "limitative result" of PL (and Logic). A problem is NP-complete when it is very difficult to be computed!

PSAT (Two Remarks)

- Remark 1: Cook's Theorem know as Cook-Levin's Theorem, since L. Levin proved an equivalent theorem two years later (1973).
- Remark 2: PSAT (i.e. SAT in PL) and Cook-Levin's Theorem motivated an active area of reseach on heuristic algorithms for PSAT.
- PSAT: a fundamental problem in applications See for instance: http://www.satisfiability.org

Davis-Putnam's Procedure

Premise

- The Davis–Putnam algorithm was developed by Martin Davis and Hilary Putnam in 1960 for checking the validity of a FOL-formula.
- The Davis–Putnam algorithm is not a decision procedure in the strict sense, as it does *not* terminate on some inputs.
- We are interested in the PL part of the DP algorithm well-known as DPLL procedure.

DPLL Procedure Introduction

- The DPLL algorithm is a decision procedure developed by M. Davis, H. Putnam, G. Logemann, and D. Loveland in 1962 for deciding the satisfiability of a proposition in conjunctive normal form (CNF).
- DPLL is a highly efficient procedure for a NP-complete and important SAT problem.
- We present the main elements of DPLL.

Background Concepts

- To understand the DPLL procedure we first require to know some background concepts:
  - literal (negative / positive)
  - clause (unit / empty)
  - conjunctive normal form (CNF)
  - the conversion procedure to rewrite a proposition to its equivalent CNF.

Literals

- A literal is either an atomic proposition or the negation of an atomic proposition, i.e., for all propositional variables P,

  P and ¬P are literals.
  - P is a positive literal, ¬P is a negative literal
- Examples: 'inLab-monkey', 'airplane', '¬highBox=1mt', '¬inLab-monkey', ...

Clauses

- A clause is the disjunction of literals.
  - Example: B∨¬C∨¬D, Being∨¬Being, ...
- A unit clause is a clause that contains only a single literal.
  - Example: B,¬C are unit clauses, B∨¬C isn't
- A empty clause is a clause with all variables assigned so that all the literals in it are false.

Conjunctive Normal Form (CNF)

- A proposition is in conjunctive normal form (CNF) if it is a (finite) conjunction of clauses.
  - Examples: $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$, $\neg A \land B$, $A \land (B \lor C \lor \neg D)$, ...
  - Counter-examples:
    $\neg(A \land B)$: $\neg$ is the outmost operator
    $A \land (B \lor C \land \neg D)$: $\land$ is nested within $\lor$

Conversion to CNF

- Theorem. Every proposition P has a CNF, i.e., P can be converted (in polynomial time) into a proposition Q such that:
  - Q is in CNF
  - P and Q are equivalent (i.e., have the same truth table).
- *Proof:* See e.g. (Mendelson, 1987) [Prop. 1.4, Ex. 1.41(a)].

Conversion to CNF (Remark)

- A conversion to a CNF that operates as in the theorem above is said to be complete.
- The algorithm implementing the conversion to a CNF is based on the tautologies about logical equivalence (or double implication):

- double-negation law (i.e. elimination of $\neg\neg$)
- De Morgan's laws
- distributive laws
- elimination of $\rightarrow$ and $\leftrightarrow$ (see the next slide)

Conversion to CNF Basic Tautologies

- $\neg(\neg\alpha) \leftrightarrow \alpha$ — double-negation elimination
- $\neg(\alpha \land \beta) \leftrightarrow (\neg\alpha \lor \neg\beta)$ — De Morgan's Law for $\land$
- $\neg(\alpha \lor \beta) \leftrightarrow (\neg\alpha \land \neg\beta)$ — De Morgan's Law for $\lor$
- $(\alpha \land (\beta \lor \gamma)) \leftrightarrow ((\alpha \land \beta) \lor (\alpha \land \gamma))$ — distributivity of $\land$ over $\lor$
- $(\alpha \lor (\beta \land \gamma)) \leftrightarrow ((\alpha \lor \beta) \land (\alpha \lor \gamma))$ — distributivity of $\lor$ over $\land$
- $(\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \lor \beta)$ — $\rightarrow$-elimination
- $(\alpha \leftrightarrow \beta) \leftrightarrow ((\alpha \rightarrow \beta) \land (\beta \rightarrow \alpha))$ — $\leftrightarrow$-elimination

Example

Conversion to CNF of '$A \leftrightarrow (B \lor C)$'

1. Eliminate $\leftrightarrow$: $(A \rightarrow (B \lor C)) \land ((B \lor C) \rightarrow A)$
2. Eliminate $\rightarrow$: $(\neg A \lor B \lor C) \land (\neg(B \lor C) \lor A)$
3. Move $\neg$ inwards using de Morgan's laws: $(\neg A \lor B \lor C) \land ((\neg B \land \neg C) \lor A)$
4. Apply distributivity of $\lor$ over $\land$ and flatten: $(\neg A \lor B \lor C) \land (\neg B \lor A) \land (\neg C \lor A)$ (*)

(*) in CNF and equivalent to '$A \leftrightarrow (B \lor C)$'.

Disjunctive Normal Form (DNF)

- Although not necessary to the DPLL procedure, we mention also the DNFs.
- Premise: A conjunctive clause is a (finite) conjunction of literals. NB: it is not a clause!
- A proposition is in disjunctive normal form (DNF) if it is a (finite) disjunction of conjunctive clauses.
- Examples: $\neg A \lor B$, $A \lor (B \land C \land \neg D)$, ...

CNFSAT-Problem

- Definition.
  CNFSAT = {P in CNF | exists ν s.t. ν |= P}.
  - Informally, CNFSAT is a set of propositions P in CNF such that there is some truth-valuation (also called truth- or propositional assignment) of the truth-values to the propositional variables in P that will make P true.
- CNFSAT-Problem: Is CNFSAT decidable?
- Like PSAT, CNFSAT is NP-complete.

DPLL Procedure

- DPLL employs a backtracking search to explore the space of propositional variables truth-valuations of an proposition P in CNF, looking for a satisfying truth-valuation of P.
- DPLL solves the CNFSAT-Problem by searching a truth-assignment that satisfies all clauses in the input proposition.

DPLL Procedure
Main Steps

- The DPLL procedure does the following:

1. chooses a literal in the input proposition P
2. assignes a truth-value (*) to its variable as to satisfy it
3. simplifies P by removing all clauses in P which become true under the truth-assignment in step 2. and all literals in P that become false from the remaining clauses
4. recursively checks if the simplified proposition obtained in step 3. is satisfiable; if this is the case, then P is satisfiable. Otherwise, the same recursive checking is done assuming the opposite truth value (*).

DPLL Procedure

- **Input**: a propositin P in CNF.
- **Output**: "P satisfiable" or " P unsatisfiable".

Step 1 : Unit Propagation
While there is no empty clause and an unit clause exists, select an unit clause and assign a variable in it to satisfy it.

Step 2 : Satisfiability Checking
If all clauses are satisfied, return "satisfiable".

Steps 3 and 4 : see the next slide.

DPLL Procedure (cont')

Step 3 : Unsatisfiability Checking
If an empty clause exists, return "unsatisfiable".

Step 4 : Splitting Rule
- Select a variable whose value is not assigned.
- Assign True to the variable and call DPLL. If the result is "satisfiable" then return "satisfiable".
- Otherwise, assign False to the variable an call DPLL again. Return the result of it.

DPLL Procedure
Example 1

- Take $P = A \wedge (A \vee \neg A) \wedge B$
- The DPLL procedure does the following:
  1. chooses a literal in P, e.g., A
  2. assignes a truth-value to A, e.g., $\nu(A) = T$
  3.1 simplifies P by removing all clauses in P which become true under $\nu(A) = T$ in step 2: remove A and $(A \vee \neg A)$
  [in fact: by assuming $\nu(A) = T$ we have $\nu(A \vee \neg A) = T$]
  3.2 simplifies P by removing all literals in P that become false from the remaining clauses: nothing to remove
  [in fact: B is the only remaining clause, and $\nu(B)$ = ?]

DPLL Procedure Example 1 (cont')

- 4. recursively checks if the simplified proposition obtained in step 3. is satisfiable:
  - simplified proposition from step 3: B
  - is B satisfiable? Yes: just define ν(B) = T, hence ν |= B.

  4.1 if this is the case, the original formula is satisfiable:
  - Yes, this is the case:
  a model for it is ν s.t. ν(A) = T and ν(B) = T;

  4.2 otherwise, the same recursive check is done assuming the opposite truth value:
  - No, not the case, DPLL did terminate in 4.1.

DPLL Procedure Example 2

- Take P = C ∧ (A ∨ ¬A) ∧ B
- The DPLL procedure does the following:
  1. chooses a literal in P, e.g., C
  2. assignes a truth-value to C, e.g., ν(C) = T
  3.1 simplifies P by removing all clauses in P which become true under ν(C) = T in step 2: remove C

  3.2 simplifies P by removing all literals in P that become false from the remaining clauses: nothing to remove
  [the remaining clauses were not assigned a truth-value]

DPLL Procedure Example 2 (cont')

- 4. recursively checks if the simplified proposition obtained in step 3. is satisfiable:
  - simplified proposition from step 3: (A∨¬A) ∧ B
  - is (A∨¬A) ∧ B satisfiable?
- Call DPLL from step 1:
  1. choose a literal in (A∨¬A) ∧ B, e.g., A
  2. assign a truth-value to A, e.g., ν(A) = T
  3.1 simplify by removing all clauses in (A∨¬A) ∧ B which become true under ν(A) = T: remove (A∨¬A)

DPLL Procedure Example 2 (cont')

- 3.2 simplify by removing all literals in (A∨¬A) ∧ B that become false from the remaining clauses: nothing to remove
- 4. recursively checks if the simplified proposition obtained in step 3. is satisfiable:
  - simplified proposition from step 3: B
  - is B satisfiable? Yes: just define ν(B) = T, hence ν |= B
  4.1 if this is the case, the original formula is satisfiable:
  - Yes, this is the case:
  a model for it is ν s.t. ν(A) = T and ν(B) = T
- DPLL terminate with output: "C∧(A∨¬A)∧B satisfiable"

Horn Form Clauses and Formulas

- A Horn clause is a clause of which *at most one* is positive.
  - Example: ¬A ∨ B ∨ ¬C ∨ D.
  - It is definite if *exactly one* literal is positive.
    - Example: ¬A ∨ ¬B ∨ ¬C ∨ D.
- P is in horn form (P Horn formula) if P is in CNF and all its clauses are all Horn clauses.

Horn Form Clauses and Formulas

- Theorem. Every Horn clause can be written as conjunction of an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
- *Proof:* Exercise.
- Example. ¬A ∨ B ∨ ¬C ∨ D can be rewritten as (A ∧ ¬B ∧ C) → D

HORNSAT-Problem

- <u>Definition.</u>
  HORNSAT = {P Horn | exists v s.t. v |= P}.
  - Informally, HORNSAT is a set of propositions P in Horn form such that there is some truth-valuation (also called truth- or propositional assignment) of the truth-values to the propositional variables in P that will make P true.
- HSAT-Problem: Is HORNSAT decidable?
- Unlike PSAT and CNFSAT (NP-complete), H-SAT is solvable in polynomial time.

Summary

Pros and Cons - Pros

- PL is declarative: syntax captures facts.
- PL allows partial/disjunctive/negated knowledge (unlike most databases).
- PL is compositional: $\models A \wedge B$ from $\models A$ and $\models B$.
- Meaning in PL is context-independent (unlike natural language).
- PSAT fundamental in important applications.

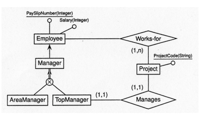
Pros and Cons - Cons

- PL has very limited expressive power! (yet useful in applications, we'll in a few slides)
  - <u>Example 1</u>: In PL you cannot say "pits cause breezes in adjacent squares" except by writing one sentence for each square.
  - <u>Example</u> 2 (De Morgan, 1864): inferences like "*All horses are animals; therefore, the head of a horse is the head of an animal*" cannot be handled in PL. Why?

Example (KB)

In PL we can suppose we know that:

1. AreaManager → Manager
2. TopManager → Manager
3. Manager → Employee
4. TopManager(John)



Observe: 1,2,3, 4 *are* propositions.

Example (cont')

But we can't <u>deduce</u> the following:

5. Manager(John)
6. Employee(John)

- Why?

1. AreaManager → Manager
2. TopManager → Manager
3. Manager → Employee
4. TopManager(John)

Example (cont')

But we can't deduce the following:

5. Manager(John)
6. Employee(John)

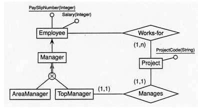
1. AreaManager → Manager
2. TopManager → Manager
3. Manager → Employee
4. TopManager(John)

- Why?
- Because of / can't be used together!
  - recall that in 'TopManager(John)', in PL 'John' is not separable from 'TopManager'.

Example (cont')

In PL we *can't* suppose we know that:

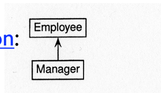
1. AreaManager(x) → Manager(x)
2. TopManager(x) → Manager(x)
3. Manager(x) → Employee(x)
4. TopManager(John)



NB: 1,2,3 are not propositions, because of variables!

Pros and Cons - Cons

- The semantics of PL - as it is based on truth valuation (i.e. True, False) does *not* help the concept modeler.
- Example 2: Meaning of IS-A relation:



  - Manager ⊆ Employee
  - **Question:** Is 'Manager→Employee' a proposition that models the schema?

Summary

- PL has not much to offer to the modeler apart from background notions and PSAT algorithms crucial in many application areas.
- Philosophically interesting question: ClassicalPL (i.e., CLP, where |= A∨¬A) or PL?
- Pros:
  1. PL is declarative: syntax captures facts
  2. Context-independent semantics