Logic Review

Overview

- Propositional logic
- Predicate calculus
- Soft logic / annotated logic

Propositional Logic

- We assume the existence of a universe of ground atomic propositions (‘atoms’ or ‘ground atoms’)
- Atoms can be either true or false

- Running example:

$$U = \{a_1, \ldots, a_n\}$$

Syntax

- The syntax specifies the language of the logic.
- The key element of the syntax is a formula.
- In propositional logic, formulas are usually created with three connectors – disjunction, conjunction, and negation

$$f \ ::= a \mid \neg f \mid f \wedge f \mid f \vee f$$

Note: sometimes it is useful to also consider “literals.” A literal is any ground atom or a negation of a ground atom.

Semantics

- Semantics allow us to add meaning to the syntax and is defined separately
- In propositional logic, the main semantic structure is a world
- A world is simple a subset of atoms
- Intuition: if an atom is a member of a world, it is considered true in that world – otherwise it is false
- Example:

$$U = \{a_1, a_2, a_3\}$$

$$W = \{\{\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$$

Satisfaction

- Satisfaction specifies the relationship between syntax and semantics
- Often, the symbol used for satisfaction is $\vDash$
- Satisfaction is defined recursively – the below is a standard definition of what is means for a world (w) to satisfy a formula.

$$
\begin{array}{ll}
if\ f = a: & w \vDash f \quad if\ a \in w \\
if\ f = \neg f': & w \vDash f \quad if\ w \nvDash f' \\
if\ f = f' \wedge f'': & w \vDash f \quad if\ w \vDash f'\ and\ w \vDash f'' \\
iff = f' \vee f'': & w \vDash f \quad if\ w \vDash f'\ or\ w \vDash f''
\end{array}
$$

Implication / Rules

- An implication (→) is what is used to establish logical rule. Often referred to as a
- However, given our previous standard definitions, an implication can be created based on the standard connectors (although other definitions are possible)

$$f \to f' \equiv f' \vee \neg f$$

Implication / Rule Terminology

$$f \rightarrow f'$$

Antecedent
(pre-condition,
body)

Consequent
(post-condition,
head)

$$f' \leftarrow f$$

(alternate notation)

Notes on Implication / Rules

- Some frameworks treat everything as a rule. They represent formulas as a rule with no body, often referring to them as “facts”:

$$f \leftarrow$$

- The intuition is that the body consists of a tautology (the body is always true)

- Rule heads are typically atoms or negations

Logic programs

- A logic program is a set of logical formulas
- Often, we use the notation Π
- It is useful to separate out *facts* from the logic program. Facts are typically non-rules (often atoms, conjunctions, or rules with no body).
  - The intuition is that a set of facts is for a given situation, while the program is a set of rules that gets applied to the facts (and is more general)
- A world w satisfies program Π if it satisfies all elements of Π

Consistency

- A program is consistent if there exists a world that satisfies it.
- An example of an inconsistent program:

$$\Pi = \{a, \neg b, a \rightarrow b\}$$

- In most cases (e.g., where there are little or no restrictions on the logic) this is NP-hard as it is equivalent to satisfiability

Entailment

- If a program entails a query formula f, it intuitively means that f can be concluded from the information in the program.
- For program otation Π, let $\mathrm{M}(\Pi)$ be the set of all satisfying worlds. Likewise, for f, let $\mathrm{M}(f)$, be the set of satisfying worlds for f.
- We say Π entails f ($\Pi \models f$) iff $\mathrm{M}(\Pi) \subseteq \mathrm{M}(f)$
- This problem is coNP hard for most logics

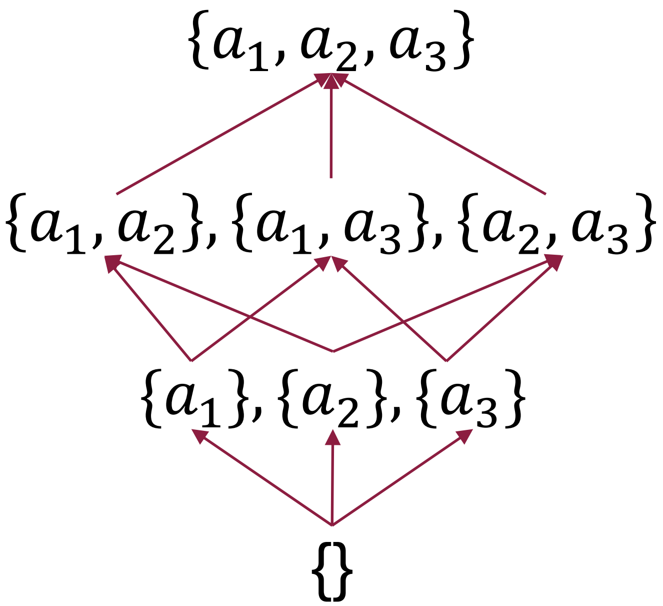
Set of Worlds as a Lattice Structure

- Lattice theory often comes in hand when dealing with logic
- In the propositional case, the set of all worlds is just the powerset of atoms.
- This forms a complete lattice under $\subseteq$

Example Lattice

$U = \{a_1, a_2, a_3\}$

$W = \{\{\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$



Restricting Logic

- Most work on logic adds some restrictions
- Lets give an example of such a logic:
  - Primary structure is a rule
  - Rules can have only a single atom in the head
  - Rules have only a conjunction of atoms in the body

- Note that not having negation greatly reduces expressiveness, but makes things a lot easier

Restricting Logic

- Given our simple logic, can we more easily compute all the atoms entailed by a program?

- Yes, and one technique to do so is to leverage lattice theory and a fixpoint operator

Fixpoint Operator

- Given a program Π, let T_Π be a function that maps worlds to worlds.
- We define it as follows:

$$T_\Pi(w) = w \cup \bigcup_{r \in \Pi} \{head(r) \; such \; that \; body(r) \subseteq w\}$$

Where for a given rule r, *head(r)* is the atom in the head and *body(r)* is the set of atoms in the conjunction of the body.

Intuitively, it says that if you have a set of atoms (a world), return that world plus any atoms that can be concluded by a single application of a rule in the program.

Example

$U = \{a_1, a_2, a_3\}$

$W = \{\{\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$

$\Pi = \{\rightarrow a_1, a_1 \rightarrow a_2, a_2 \rightarrow a_3\}$

$T_\Pi(\{a_2\}) = \{a_1, a_2, a_3\}$

$T_\Pi(\{a_3\}) = \{a_1, a_3\}$

$T_\Pi(\{\}) = \{a_1\}$

Multiple Applications of the Fixpoint Operator

- It is useful to apply the fixpoint operator multiple times
- We can define that as follows:

$$T_\Pi{}^{(i)}(w) = \begin{cases} T_\Pi(w) \ \ if\ i = 1 \\ T_\Pi\left(T_\Pi{}^{(i-1)}(w)\right) \ \ if\ i > 1 \end{cases}$$

The operator has a fixed point if there exists i such that: $T_\Pi{}^{(i+1)}(w) = T_\Pi{}^{(i)}(w)$

Example

$U = \{a_1, a_2, a_3\}$

$W = \{\{\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}\}$

$\Pi = \{\rightarrow a_1, a_1 \rightarrow a_2, a_2 \rightarrow a_3\}$

$T_\Pi{}^{(1)}(\{\}) = T_\Pi(\{\}) = \{a_1\}$

$T_\Pi{}^{(2)}(\{\}) = T_\Pi(T_\Pi(\{\})) = \{a_1, a_2\}$

$T_\Pi{}^{(3)}(\{\}) = T_\Pi(T_\Pi(T_\Pi(\{\}))) = \{a_1, a_2, a_3\}$

$T_\Pi{}^{(4)}(\{\}) = T_\Pi(T_\Pi(T_\Pi(T_\Pi(\{\})))) = \{a_1, a_2, a_3\}$

Why do we care if a fixpoint exists?

- Any time we apply the T operator, we get some information on which elements of the program led to that conclusion

$$\Pi = \{\to a_1, a_1 \to a_2, a_2 \to a_3\}$$

$${T_\Pi}^{(1)}(\{\}) = T_\Pi(\{\}) = \{a_1\}$$

$\to a_1$

$${T_\Pi}^{(2)}(\{\}) = T_\Pi(T_\Pi(\{\})) = \{a_1, a_2\}$$

$a_1 \to a_2$

$${T_\Pi}^{(3)}(\{\}) = T_\Pi(T_\Pi(T_\Pi(\{\}))) = \{a_1, a_2, a_3\}$$

$a_2 \to a_3$

Why do we care if a fixpoint exists?

- The fact that there is a logical connection utilized in an application allows us to prove that any world that satisfies the program is a superset of the least fixed point – this is called a ***minimal model***
- This implies we know $\mathrm{M}(\Pi)$ and entailment becomes much easier
- So the two items that have to be proven is that
  - (1.) the fixed point exits
  - and (2.) it corresponds with a minimal model

Proving the Fixed Point Exists

- By the Tarski-Knaster fixpoint theorem, any monotonic function over a complete lattice is guaranteed a fixed point.

- In the proof, we have to first prove that the structure is a complete lattice
  - This is trivial for the power set, as it has clear top and bottom element – but in some work it is not straight-forward

- Then we have to show that the function is monotonic.
  - In our example, this is easy, as every application of the operator includes the argument as a subset by definition

Predicate Calculus

- Predicate calculus can be thought of as a way to specify the atomic propositions
- The key components are:
  - Constants
  - Variable symbols
  - Predicate symbols

Predicate + Constant(s) = (Ground) atomic proposition

Predicate + Variable symbol(s) = Non-ground atomic proposition

Non-ground atoms are the key item that differentiates Predicate Calculus from Propositional Calculus

(Predicate Calculus is also called “First Order Logic” (FOL))

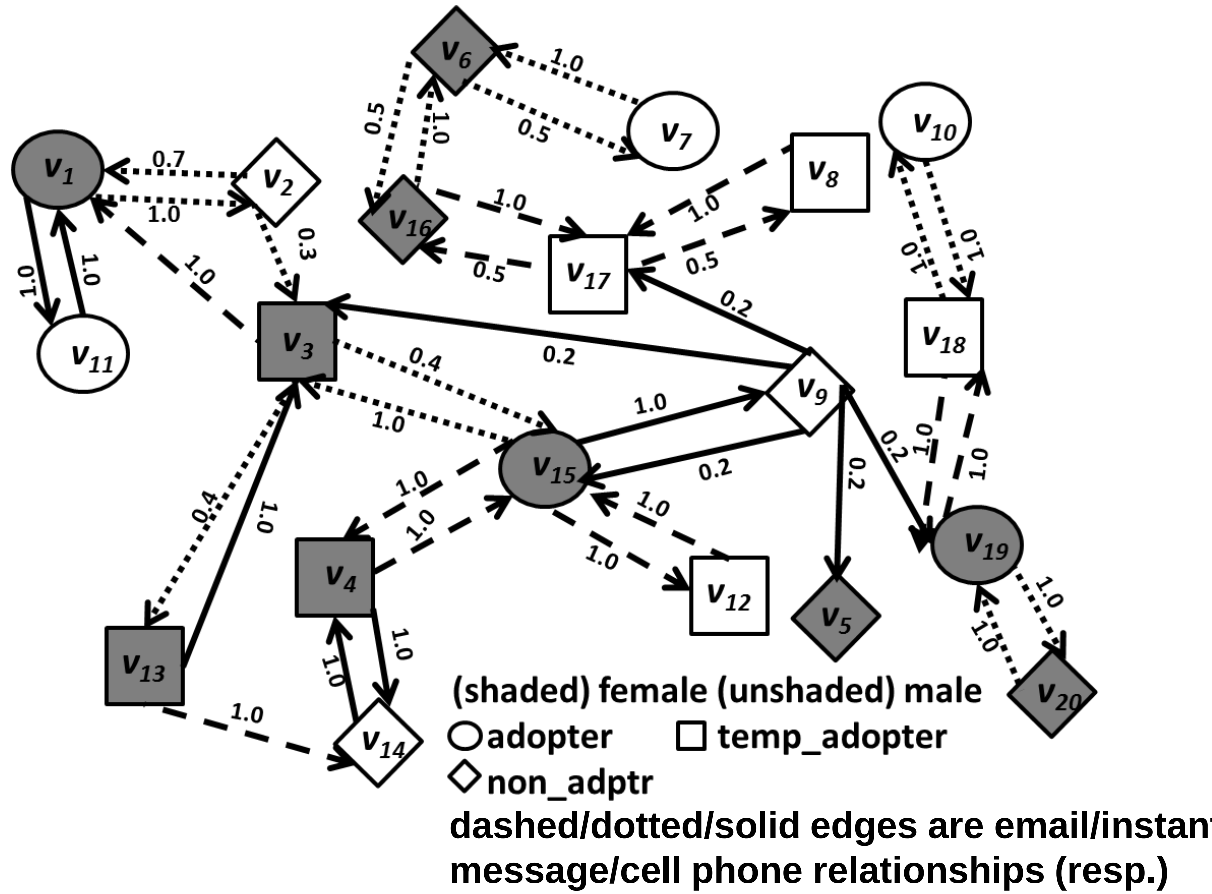
Example

- Let $\mathrm{C} = \{v_1, v_2, \dots, v_{20}\}$ be a set of constant symbols (representing people)
- Predicates can have an arity, specifying how many arguments it can have
  - Let the set of binary predicates (2 arguments) be: $\{email\_conn, sms\_conn, cell\_conn\}$
  - Let the set of unary predicates (1 argument) be: $\{male, female, adopter, non\_adopter, temp\_adopter\}$
- Example ground atoms:

$$female(v_1),\ adopter(v_{11}),\ cell\_conn(v_1, v_{11})$$

Predicate Calculus and Knowledge Graphs

If we restrict our language to only binary and unary predicates, we can treat a knowledge base of facts as graph.



Variable Symbols

- Variable symbols allow us to talk about a set of atoms that share a predicate.
- Example:
  - Non ground atom *adopter(X)* ground to:
    - $adopter(v_1)$, $adopter(v_7)$, $adopter(v_{10})$, $adopter(v_{11})$, $adopter(v_{15})$, $adopter(v_{19})$

Quantifiers

- Universal
  - $\forall\, X : p(X) = \bigwedge_{v \in C} p(v)$
  - For all X, $p(X)$ is true
- Existential
  - $\exists\, X : p(X) = \bigvee_{v \in C} p(v)$
  - There exists some X such that $p(X)$ is true

Grounding

- The use of quantifiers and variable symbols can lead to grounding – which is transforming non-ground atoms to ground atoms
- Existential quantifiers often appear in queries (e.g., does the logic program entail an existentially quantified formula) – this leads to many queries being performed (to prove existence). Think of it as a large disjunction.
- Universal quantifiers have a similar issue on the query side, but also tend to appear in non-ground programs. This can cause the grounded version of the program to become many times larger.

Annotated Logic

- There are several frameworks that “annotate” logical syntax with additional information – they use special semantic structures

Annotation Type	Examples	Semantic Structure
Scalar in [0,1]	Fuzzy logic VanEmden logic	Interpretation that maps atoms to reals
Scalar as an interval subset of [0,1]	MANCALog Real valued logic of LNN’s	Interpretation that maps atoms to reals
Point Probability	PCTL tp-programs	MDP
Probability interval	Nilsson logic AP-programs APT logic	PDF over worlds
Elements of a lattice structure	Generalized annotated programs	Interpretation that maps atoms to elements of the lattice

Fuzzy Operators

- The fuzzy operators are used as functions to provide the feeling of various logical operations
- T-Norms: Fuzzy conjunction
- T-Conorms: Fuzzy disjunction
- Strong negation: 1-x (where x is the value associated with the atom)
- Aggregate operators: Quantification (for predicate calculus)

Fuzzy Operators

Table 1
The t-norms of interest.

Name	T-norm	Properties
Gödel (minimum)	$T_G(a,b) = \min(a,b)$	idempotent, continuous
Product	$T_P(a,b) = a \cdot b$	strict
Łukasiewicz	$T_{LK}(a,b) = \max(a+b-1,0)$	continuous
Drastic product	$T_D(a,b) = \begin{cases} \min(a,b), & \text{if } a=1 \text{ or } b=1 \\ 0, & \text{otherwise} \end{cases}$	
Nilpotent minimum	$T_{nM}(a,b) = \begin{cases} 0, & \text{if } a+b \leq 1 \\ \min(a,b), & \text{otherwise} \end{cases}$	left-continuous
Yager	$T_Y(a,b) = \max(1-((1-a)^p + (1-b)^p)^{\frac{1}{p}}, 0), p \geq 1$	continuous

Table 2
The t-conorms of interest.

Name	T-conorm	Properties
Gödel (maximum)	$S_G(a,b) = \max(a,b)$	idempotent, continuous
Product (probabilistic sum)	$S_P(a,b) = a+b-a \cdot b$	strict
Łukasiewicz	$S_{LK}(a,b) = \min(a+b,1)$	continuous
Drastic sum	$S_D(a,b) = \begin{cases} \max(a,b), & \text{if } a=0 \text{ or } b=0 \\ 1, & \text{otherwise} \end{cases}$	
Nilpotent maximum	$S_{nM}(a,b) = \begin{cases} 1, & \text{if } a+b \geq 1 \\ \max(a,b), & \text{otherwise} \end{cases}$	right-continuous
Yager	$S_Y(a,b) = \min((a^p + b^p)^{\frac{1}{p}}, 1), p \geq 1$	continuous

Table 3
Some common aggregation operators.

Name	Generalizes	Aggregation operator
Minimum	T_G	$A_{T_G}(x_1, \ldots, x_n) = \min(x_1, \ldots, x_n)$
Product	T_P	$A_{T_P}(x_1, \ldots, x_n) = \prod_{i=1}^n x_i$
Łukasiewicz	T_{LK}	$A_{T_{LK}}(x_1, \ldots, x_n) = \max(\sum_{i=1}^n x_i - (n-1), 0)$
Maximum	S_G	$E_{S_G}(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$
Probabilistic sum	S_G	$E_{S_P}(x_1, \ldots, x_n) = 1 - \prod_{i=1}^n (1-x_i)$
Bounded sum	S_{LK}	$E_{S_{LK}}(x_1, \ldots, x_n) = \min\left(\sum_{i=1}^n x_i, 1\right)$

From van Krieken et al., *AIJ* 2022.

ASU®
Ira A. Fulton Schools of
Engineering
Arizona State University