Logical Foundations

CS3100 Fall 2019

Review

Previously

- Prolog basics

This lecture

- Logical foundations of prolog
  - First-order logic
    - Syntax, Semantics and properties
  - Definite Clause programs
    - Syntax, connection to prolog, SLD resolution

First-order logic

Terms and functions:

$$\begin{array}{rcl} \text{term} & := & \text{constant} \mid \text{variable} \mid \text{functions} \\ \text{functions} & := & f(t1, t2, \ldots, tn) \mid g(t1, t2, \ldots., tn) \\ & & \quad \text{where f and g are function symbols.} \\ & & \quad \text{where t1,t2... are terms.} \end{array}$$

Natural numbers

Consider the terms for encoding natural numbers $\mathbb{N}$.

- **Constant:** Let z be 0.
- **Functions**: Given the natural numbers x and y, let the function
  - $\mathrm{s}(x)$ represent the successor of x
  - $\mathrm{mul}(x, y)$ represent the product of x and y.

- $\text{square}(x)$ represent the square of x.

First-order logic

$$
\begin{array}{rcll}
t \in \text{term} & := & \text{constant} \mid \text{variable} \mid \text{functions} & \\
 & & & \\
f, g \in \text{formulas} & := & p(t_1, \ldots, t_n) & \text{where } p \text{ is the predicate symbol} \\
 & \mid & \neg f \mid f \land g \mid f \lor g \mid f \to g \mid f \leftrightarrow g & \\
 & \mid & \forall X.\, f \mid \exists X.\, f & \text{where } X \text{ is a variable}
\end{array}
$$

Predicates on natural numbers

- $\text{even}(x)$ - the natural number x is even.
- $\text{odd}(x)$ - the natural number x is odd.
- $\text{prime}(x)$ - the natural number x is prime.
- $\text{divides}(x, y)$ - the natural number x divides y.
- $\text{le}(x, y)$ - the natural number x is less than or equal to y
- $\text{gt}(x, y)$ - the natural number x is greater than y.

Precedence

From strongest to weakest

1. $\neg$
2. $\lor$
3. $\land$
4. $\to, \leftrightarrow$
5. $\forall, \exists$

Precedence

Hence,

$$((\neg b) \land c) \to a)$$

can be simplified to

$$\neg b \land c \to a$$

Some statements on natural numbers

- Every natural number is even or odd, but not both.
- A natural number is even if and only if it is divisible by two.
- If some natural number, x, is even, then so is x^2.
- A natural number x is even if and only if $x + 1$ is odd.
- Any prime number that is greater than 2 is odd.
- For any three natural numbers x, y, and z, if x divides y and y divides z, then x divides z.
- There exists an odd composite number (recall, composite number is greater than 1 and not prime).
- Every natural number greater than one has a prime divisor.

Some statements on natural numbers

- Every natural number is even or odd, but not both.
  - $\forall x. ((\mathrm{even}(x) \ \vee \ \mathrm{odd}(x)) \ \wedge \ \neg(\mathrm{even}(x) \ \wedge \ \mathrm{odd}(x)))$
- A natural number is even if and only if it is divisible by two.
  - $\forall x. \mathrm{even}(x) \leftrightarrow \mathrm{divides}(2, x)$
- If some natural number, x, is even, then so is x^2.
  - $\forall x. \mathrm{even}(x) \rightarrow \mathrm{even}(\mathrm{square}(x))$

Some statements on natural numbers

- A natural number x is even if and only if $x + 1$ is odd.
  - $\forall x. \mathrm{even}(x) \leftrightarrow \mathrm{odd}(\mathrm{s}(x))$
- Any prime number that is greater than 2 is odd.
  - $\forall x. \mathrm{prime}(x) \ \wedge \ \mathrm{gt}(x, \mathrm{s}(\mathrm{s}(z))) \rightarrow \mathrm{odd}(x)$
- For any three natural numbers x, y, and z, if x divides y and y divides z, then x divides z.
  - $\forall x, y, z. \mathrm{divides}(x, y) \ \wedge \ \mathrm{divides}(y, z) \rightarrow \mathrm{divides}(x, z)$

Some statements on natural numbers.

- There exists an odd composite number.
  - $\exists x. \mathrm{odd}(x) \ \wedge \ \mathrm{composite}(x)$
- Every natural number greater than one has a prime divisor.
  - $\forall x. \mathrm{gt}(x, \mathrm{s}(z)) \rightarrow (\exists p. \mathrm{prime}(p) \ \wedge \ \mathrm{divides}(p, x))$

Logical Equivalences

$$
\begin{array}{rcl}
\neg\neg f & \equiv & f \\
f \rightarrow g & \equiv & \neg f \vee g \\
f \leftrightarrow g & \equiv & (f \rightarrow g) \wedge (g \rightarrow f) \\
\neg(f \vee g) & \equiv & \neg f \wedge \neg g \\
\neg(f \wedge g) & \equiv & \neg f \vee \neg g \\
\neg\forall x.\, f(x) & \equiv & \exists x. \neg f(x) \\
\neg\exists x.\, f(x) & \equiv & \forall x. \neg f(x)
\end{array}
$$

Logical Equivalences

$$
\begin{array}{rcl}
\forall x.\, (f(x) \wedge g(x)) & \equiv & (\forall x.\, f(x)) \wedge (\forall x.\, g(x)) \\
\forall x.\, (f(x) \vee g(x)) & \not\equiv & (\forall x.\, f(x)) \vee (\forall x.\, g(x))
\end{array}
$$

Pick f as $even$ and g as odd.

$$
\begin{array}{rcl}
\exists x.\, (f(x) \vee g(x)) & \equiv & (\exists x.\, f(x)) \vee (\exists x.\, g(x)) \\
\exists x.\, (f(x) \wedge g(x)) & \not\equiv & (\exists x.\, f(x)) \wedge (\exists x.\, g(x))
\end{array}
$$

Pick f as $even$ and g as odd.

Inference rules

$$
\frac{f \quad f \rightarrow g}{g} \ (\rightarrow E) \qquad \frac{\forall x.\, f(x)}{f(t)} \ (\forall E)
$$

$$
\frac{f(t)}{\exists x.\, f(x)} \ (\exists I) \qquad \frac{f \quad g}{f \wedge g} \ (\wedge I)
$$

Interpretation

- What we have seen so far is a syntactic study of first-order logic.
  - Semantics = meaning of first-order logic formulas.
- Given an alphabet A from which terms are drawn from and a domain $\mathcal{D}$, an **interpretation** maps:
  - each constant $c \in A$ to an element in $\mathcal{D}$
  - each n-ary function $f \in A$ to a function $\mathcal{D}^n \rightarrow \mathcal{D}$
  - each n-ary preducate $p \in A$ to a relation $D_1 \times \ldots \times D_n$

Interpretation

For our running example, choose the domain of natural numbers $\mathbb{N}$ with

- The constant z maps to 0.
- The function $\mathrm{s}(x)$ maps to the function $\mathrm{s}(x) = x + 1$
- The predicate le maps to the relation $\leq$

Models

- A **model** for a set of first-order logic formulas is equivalent to the assignment to truth variables in predicate logic.
- A interpretation M for a set of first-order logic formulas P is a model for P iff every formula of P is true in M.
- If M is a model for f, we write $M \vDash f$, which is read as "models" or "satisfies".

Models

Take $f = \forall y.\, \mathrm{le}(z, y)$. The following are models for f

- Domain $\mathbb{N}$, z maps to 0, $\mathrm{s}(x)$ maps to $\mathrm{s}(x) = x + 1$ and le maps to $\leq$.
- Domain $\mathbb{N}$, z maps to 0, $\mathrm{s}(x)$ maps to $\mathrm{s}(x) = x + 2$ and le maps to $\leq$.
- Domain $\mathbb{N}$, z maps to 0, $\mathrm{s}(x)$ maps to $\mathrm{s}(x) = x$ and le maps to $\leq$.

whereas the following aren't:

- The integer domain $\mathbb{Z}, \ldots$
- Domain $\mathbb{N}$, z maps to 0, $\mathrm{s}(x)$ maps to $\mathrm{s}(x) = x + 1$ and le maps to $\geq$

Quiz

Which of these interpretations are models of $f = \forall y.\, le(z, y)$?

1. Domain $\mathbb{N}$, z maps to 1, $s(x)$ maps to $s(x) = x + 1$ and le maps to $\leq$.
2. Domain $\mathbb{N}$, z maps to 1, $s(x)$ maps to $s(x) = x * 2$ and le maps to $\leq$.
3. Domain $\mathbb{N}$, z maps to 0, $s(x)$ maps to $s(x) = x + 1$ and le maps to $<$.
4. Domain is the domain of sets, z maps to $\emptyset$, $s(x)$ maps to $s(x) = \{x\}$ and $le(x, y) = x \subseteq y \vee \exists e \in y.\, le(x, e)$.

Quiz

Which of these interpretations are models of $f = \forall y. le(z, y)$?

1. Domain $\mathbb{N}$, z maps to 1, $s(x)$ maps to $s(x) = x + 1$ and le maps to $\leq$. **yes**
2. Domain $\mathbb{N}$, z maps to 1, $s(x)$ maps to $s(x) = x * 2$ and le maps to $\leq$. **yes**
3. Domain $\mathbb{N}$, z maps to 0, $s(x)$ maps to $s(x) = x + 1$ and le maps to $<$. **no**
4. Domain is the domain of sets, z maps to $\emptyset$, $s(x)$ maps to $s(x) = \{x\}$ and $le(x, y) = x \subseteq y \vee \exists e \in y. le(x, e)$. **yes**

Models

- A set of forumulas P is said to be **satisfiable** if there is a model M for P.
- Some formulas do not have models. Easiest one is $f \wedge \neg f$
  - Such (set of) formulas are said to be **unsatisfiable.**

Logical consequence & validity

Given a set of formulas P, a formula f is said to be a logical consequence of P iff for every model M of P, $M \vDash f$.

How can you prove this?

- Show that $\neg f$ is false in every model M of P.
  - Equivalent to, $P \cup \neg f$ is **unsatisfiable.**

A formula f is said to be **valid**, if it is true in every model (written as $\vDash f$).

Theorem: It is undecidable whether a given first-order logic formula f is **valid.**

Restricting the language

- Clearly, the full first-order logic is not a practical model for computation as it is undecidable.
  - How can we do better?
- Restrict the language such that the language is **semi-decidable.**
- A language L is said to be **decidable** if there exists a turing machine that
  - accepts every string in L and
  - rejects every string not in L
- A language L is said to be **semi-decidable** if there exists a turing machine that

- accepts every string in L and
- for every string not in L, rejects it or loops forever.

Definite logic programs

- Definite clauses are such a restriction on first-order logic that is semi-decidable.
- Prolog is basically programming with definite clauses.
- In order to define definite clauses formally, we need some auxiliary definitions.

Definite clauses

- An **atomic forumla** is a formula without connectives.
  - $\mathrm{even}(x)$ and $\mathrm{prime}(x)$
  - but not $\neg\mathrm{even}(x)$, $\mathrm{even}(x) \vee \mathrm{prime}(y)$
- A **clause** is a first-order logic formula of the form $\forall(L_1 \vee \ldots \vee L_n)$, where every L_i is an atomic formula (a postive literal) or the negation of an atomic formula (a negative literal).
- A **definite clause** is a clause with exactly one positive literal.
  - $\forall(A_0 \vee \neg A_1 \ldots \vee \neg A_n)$
  - Usually written down as, $A_0 \leftarrow A_1 \wedge \ldots \wedge A_n$, for $n \geq 0$.
  - or more simply, $A_0 \leftarrow A_1, \ldots, A_n$, for $n \geq 0$.
- A **definite program** is a finite set of definite clauses.

Definite Clauses and Prolog

- Prolog facts are definite clauses with no negative literals.
  - The prolog fact `even(z)` is equivalent to
  - the definite clause $\forall z.\, \mathrm{even}(z) \leftarrow \top$, where $\top$ stands for true.
- Prolog rules are definite clauses.
  - The prolog rule `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)` is equivalent to
  - the definite clause $\forall x, y, z.\, \mathrm{ancestor}(x, y) \leftarrow \mathrm{parent}(x, z) \ \wedge \ \mathrm{ancestor}(z, y)$
  - equivalent to, $\forall x, y.\, \mathrm{ancestor}(x, y) \leftarrow \exists z.\, \mathrm{parent}(x, z) \ \wedge \ \mathrm{ancestor}(z, y)$

Consistency of Definite Clause Programs

- Every definite clause program has a model!
- Proof
  - there is no way to encode negative information in definite clause programs.

- there is no way to encode negative information in definite clause programs.
  - Hence, there is no way to construct an inconsistent system (such as $f \wedge \neg f$).
  - Therefore, every definite clause program has a model.

Prolog Queries

- Let us assume that the prolog program P is family tree of House Stark encoded in the previous lecture.
- We would like to answer "is Rickard the ancestor of Robb?"
  - $q = \text{ancestor}(rickard, robb)$
- We construct a logical statement
  - $\neg\text{ancestor}(rickard, robb)$
  - which is the **negation** of the original question.

Prolog Queries

- The system attempts to show that $\neg\text{ancestor}(rickard, robb)$ is false in every model of P.
  - equivalent to showing $P \cup \{\neg\text{ancestor}(rickard, robb)\}$ is unsatisfiable.
- Then, we can conclude that for every model M of P, $M \vDash q$.
  - that is, "Rickard is the ancestor of Robb".

SLD Resolution

- The whole point of restricting the first-order logic language to definite clauses is to have a better decision procedue.
- There is a **semi-decidable** decision procedure for definite clauses called **SLD resolution**.
  - SLD = Selective Linear Resolution with Definite Clauses.
  - given an unsatisfiable set of formulae it is guaranteed to derive false
  - however given a satisfiable set, it may never terminate.

SLD Resolution example

```
father(rickard,ned).
father(rickard,brandon).
father(rickard,lyanna).
father(ned,robb).
father(ned,sansa).
father(ned,arya).
parent(X,Y) :- father(X,Y).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
?- ancestor(rickard, robb).
```

SLD Resolution example

- The logical version goal is $\neg$ancestor(rickard,robb).
- The system attemps to disprove this by **finding a counter-example.**
  - How can I derive `ancestor(rickard,robb)` ?
- I can see a rule `ancestor(X,Y) :- parent(X,Y)` which allows me to derive `ancestor(X,Y)`.
  - the logical equivalent is, $\forall x, y. (ancestor(x, y) \leftarrow parent(x, y))$.
- **Deduce**:
  - Apply $(\forall E)$ rule for x and y and pick $x = rickard$ and $y = robb$.
  - Apply $(\rightarrow E)$ rule on the result to get a new goal $parent(rickard, robb)$.
- The original goal to derive `ancestor(rickard,robb)` has been replaced by the goal to derive `parent(rickard,robb)`.

SLD Resolution example

- How can you derive `parent(rickard,robb)` ?
- Observe the rule `parent(X,Y) :- father(X,Y)`
  - logical equivalent is $\forall x, y. parent(x, y) \leftarrow father(x, y)$.
- **Deduce**: Apply rules $(\forall E)$ and $(\rightarrow E)$.
- New goal: `father(rickard,robb)`.
- No fact matches this goal!
  - **Backtrack!**

SLD Resolution example

- How can I derive `ancestor(rickard, robb)` ?
- Observe the rule `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)`

- logical equivalent is $\forall x, y.\ ancestor(x, y) \leftarrow \exists z.\ parent(x, z) \wedge ancestor(z, y)$
- **Deduce**: Apply rules $(\forall E)$, $(\rightarrow E)$, $(\exists I)$, $(\wedge I)$ in that order.
- We get two new goals, `parent(rickard,Z)` and `ancestor(Z,robb)` where `Z` is the same variable introduced by $(\exists I)$.

SLD Resolution example

- The goal `parent(rickard,Z)` in turn leads to the goal `father(rickard,Z)`.
  - The first rule `father(rickard,ned)` unifies with this goal with `Z = ned`.
  - Hence, the first goal is proved.
- The other goal is now specialised to `ancestor(ned,robb)`.
- The second goal can now be proved as `ancestor(ned,robb)` ← `parent(ned,robb)` ← `father(ned,robb)`.
  - We have a fact `father(ned,robb)`. Hence, proved.

SLD Resolution example

- By deriving `q = ancestor(rickard,robb)` from the given program P, we have shown that $P \cup \{\neg q\}$ is unsatisfiable.
- Hence, `ancestor(rickard,robb)` is a logical consequence of the given program P.

Computation is deduction

- When a prolog program computes the result of the query, it is performing logical deduction through SLD resolution.
- In our example,
  - We picked the clauses in the order they appear in the program
  - Did a depth-first search for proof
  - Given the conjunction of goals $g1 \wedge g2$, chose to prove $g1$ first.
- SWI-Prolog implementation has the same behaviour
  - Other prolog implementation may choose different strategies BFS instead of DFS, pick last conjunct in a conjunction of goals, etc.

Tracing in SWI-Prolog

```
father(rickard,ned).
father(rickard,brandon).
father(rickard,lyanna).
father(ned,robb).
father(ned,sansa).
father(ned,arya).
parent(X,Y) :- father(X,Y).
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
?- ancestor(rickard, robb).
```

Fin.