Module 2: First Order Logic

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Learning Objectives

After successfully completing the module, you will be able to do the following:

1. Differentiate among various logics.
2. Express first order logic (FOL) expressions.
3. Differentiate syntax vs. semantics of FOL.
4. Assess challenges of inference.

Module 2 Study Guide and Deliverables

Module Theme:	Constraint Satisfaction; Reasoning in First-Order Logic
Readings:	• Module 2 online content • Russell & Norvig Chapter 6 (Constraint Satisfaction Problems), concentrate on Section 6.1 • Russell & Norvig Chapter 2 (Intelligent Agents), concentrate on Section 2.1 • Russell & Norvig Chapter 7 (Logical Agents), concentrate on Section 7.1 • Russell & Norvig Chapter 8 (First-Order Logic), concentrate on Section 8.1 • Russell & Norvig Chapter 9 (Inference in First-Order Logic), concentrate on Section 9.1
Assignments:	• Lab 2 due Sunday, September 19, at 6:00 AM ET • Assignment 2, due Wednesday, September 22, at 6:00 AM ET
Live Classrooms:	• Wednesday, September 15, from 8:00 PM to 9:00 PM ET • Thursday, September 16, from 8:00 PM to 9:00 PM ET • Live Office: Wednesday and Thursday after Live Classroom, for as long as there are questions

Loading [Contrib]/a11y/accessibility-menu.js

Logics: Various Logics

Formal Languages and Their Ontological and Epistemological Commitment

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0,1]$
Fuzzy logic	facts with degree of truth $\in [0,1]$	known interval value
Source: Russell & Norvig, Figure 8.1		

Propositional logic is the simplest. It states facts—statements with one of three values: True, False, and Unknown (or "Neither").

Despite its simplicity, this introduces an AI concept that is sometimes used: the ***Closed World Assumption***, which states that if a statement is unknown, then it is assumed to be false. For example:

p= A helicopter has flown on Mars

The most common logic is ***First-order logic (FLO)***, in which we reason at the lowest meaningful level.

Temporal logic accounts for time, as in the sentence *I used to distrust every authority and I have recently changed my attitude towards authority*—from which we can conclude *I don't distrust every authority*.

We will discuss fuzzy logic later in this module.

Syntax of First Order Logic

First Order Logic is the simplest level of logic that allows reasoning. First, we discuss its syntax.

FOL is a symbolic system that mirrors aspects of the real world. It consists of the symbols shown. Predicates take on the values `true` or `false`. They may have parameters. Think of them as you would in conversation rather than as in a programming language; for example you might say *I'm at the beach* (which is either true or false) or `AtBeach()`, a predicate with parameter—so `AtBeach`(Eric) is again true or false.

Elements of FOL

Constants: `JaneRMuldoon7, Denver, 345643, …`

Predicates: `IsProgrammer, LivesInBoston, …`

Variables: $\quad{x, y,\ldots}$

Connectives: $\land,\lor,\lnot\ldots\quad$ $\land$ is *and*; $\lor$ is *or*; $\lnot$ is *not*

Equality: $\qquad{=}\qquad\qquad$ means *is equivalent to*

Quantifiers: $\forall, \exists, \ldots\quad$ $\forall$ is *for every*; $\exists$ is *there exists*

The example below shows the beginning of a FOL representation of a real-world predicate (a statement with `true/false` value). It says `something exists which is a customer of Amazon and ….`

There may be several ways to say the same thing, such as `ACustomer(y, AmazonCo)`… but the important thing is to be consistent.

Example

English:

“Not every Amazon customer has written a review”

FOL:

$\exists{x}\ni{[\underbrace{\text{Customer(x, Amazon)}}_{\text{A predicate with 2 parameters }}\land}$

…

The following completes the FOL representation as `an x exists which is an Amazon customer and which is such that, for every p sold by Amazon, x has not reviewed p.`

Example (continued)

“Not every Amazon customer has written a review”

FOL:

\(\exists{x}\bbox[5px,border:2px solid red]{\ni}{[\text{Customer}(x, \text{Amazon})\land}\)

\(\quad\quad\,\,[\forall{p}\text{ SoldBy}(p, \text{Amazon}):\,!\text{ HasReceivewed}(x,p)]\)

Note: \(\ni\) means “such that”

The following summary begins to specify the form of FOL statements (“sentences”). It says that “every FOL sentence is either an AtomicSentence or a ComplexSentence; AtomicSentence’s are either …”.

Then shows the entire syntax (although there are variations). The success of FOL in accomplishing useful tasks is analogous to that of mathematics, and the types of logics are analogous to the types of mathematics.

Summary

\(\begin{align} Sentence &\rightarrow AtomicSentence \,|\, ComplexSentence\\ AtomicSentence &\rightarrow Predicate\, | \,Predicate(Term,\ldots)\,|\,Term=Term\\ ComplexSentence&\rightarrow (Sentence)\,|\, [Sentence]\\ & \;\,|\quad{\lnot}\,Sentence\\ & \;\,|\quad{Sentence}\, {\land}\,Sentence\\ & \;\,|\quad{Sentence}\, {\lor}\,Sentence\\ & \;\,|\quad{Sentence}\, {\Rightarrow}\,Sentence\\ & \;\,|\quad{Sentence}\, {\Leftrightarrow}\,Sentence\\ & \;\,|\quad{Quantifier}\; {Variable,}\ldots\,Sentence\\ \\ Term & \rightarrow{Function(Term, \ldots)}\\ & \;\,|\quad{Constant}\\ & \;\,|\quad{Variable}\\ \\ Quantifier & \rightarrow{\forall}\,|\,{\exists}\\ Constant & \rightarrow{A}\,|\,X_1\,|\,John\,|\,\ldots\\ Variable & \rightarrow{a}\,|\,x\,|\,s\,|\,\ldots\\ Predicate & \rightarrow{True}\,|\,False\,|\,After\,|\,Loves\,|\,Raining\,|\,\ldots\\ Function & \rightarrow{Mother}\,|\,LeftLeg\,|\,\ldots\\ Operator Precedence & \;\,:\quad{\lnot,=,\land,\lor,\Rightarrow,\Leftrightarrow} \end{align}\)

Source: Russell & Norvig

Examples

Let's look at one example: \(\lnot{\text{Brother (Tom,John)}}\)

What does the example say?

The example says that Tom is not John’s brother. This assumes that we interpret the predicates in the self-evident manner. For example, in FOL, “Brother” is just a string: it is up to us to interpret it in the real world. We do in a consistent manner, throughout the problem we are dealing with.

Loading [Contrib]/a11y/accessibility-menu.js

Test Yourself

What do the following examples say?

1. $\text{Brother(Richard,John)}\land\text{Brother(John,Richard)}$
2. $\text{King(Richard)}\lor\text{King(John)}$
3. $\lnot\text{King(Richard)}\Rightarrow\text{King(John)}$

Answer:

1. Richard and John are brothers.

2. Richard and John are kings.

3. If Richard is not a king than John is a king.

Logic is applicable to a very wide range of problems, such as the following one.

More Realistic Example

Next step in my continuing education.

Constants *BU, Northeastern, Harvard, JohnDoe …*

Predicates *LikesTo(…), Pays(…), Advancement(…)*

Variables *student, job, university, program, …*

Some Relationships Between $\forall$ and $\exists$

There is an algebra (set of rules for FOL), which includes the above, and which reflect the real world. For example, at the bottom left, saying:

some x satisfies property P

is the same as saying

it is not true that (every x satisfies P false).

$$\begin{align} \forall_x\;\lnot{P}&\equiv{\lnot\exists_x\;P}\\ \lnot\forall_x\;{P}&\equiv{\exists_x\;\lnot{P}}\\ \forall_x\;\;{P}&\equiv{\lnot\exists_x\;\lnot{P}}\\ \exists_x\;\;{P}&\equiv{\lnot\forall_x\;\lnot{P}}\\ \end{align}$$

$$\begin{align} \lnot{(P\;\lor\;{Q})}&\equiv{\lnot{P}\;\land\;\lnot{Q}}\\ \lnot{(P\;\land\;{Q})}&\equiv{\lnot{P}\;\lor\;\lnot{Q}}\\ P\;\land\;{Q}&\equiv{\lnot(\lnot{P}\;\lor\;\lnot{Q})}\\ P\;\lor\;{Q}&\equiv{\lnot(\lnot{P}\;\land\;\lnot{Q})}\\ \end{align}$$

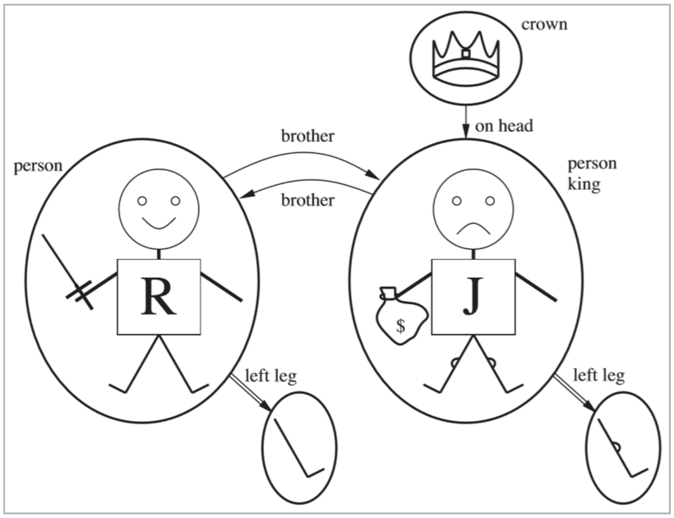
Loading [Contrib]/a11y/accessibility-menu.js

Semantics of FOL

Models are the link between (the symbolic) FOL and the situation (the possible "world") that you want to apply it to.

Each "model" links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined.

The ***domain of a model*** is the set of objects or domain elements it contains.



Source: Adapted from Russell & Norvig, p. 290.

Predicates are true or false in a world ("model").

For example, FOL:

\(\exists{x} \ni{\text{Customer}(x,y)}\land\)

\(\quad\quad\quad[\exists{y},p \ni{[\text{hasReviewed}(x,p)\land{\text{SoldBy}(p,y)}]]}\)

It says it is true in the world where \(y\) is *Amazon* or *WalMart*.

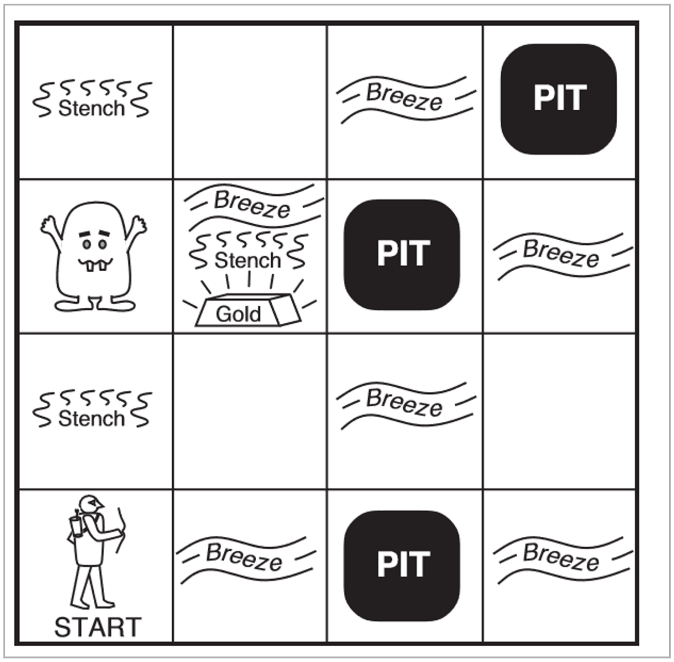
Examples

Toy Agent and FOL: Wumpus World (R&N)

Wumpus World

Loading [Contrib]/a11y/accessibility-menu.js

To exercise first order logic, Russell and Norvig developed a simplified world in which an agent moves so as to land on gold. The agent uses



FOL to make decisions about movement. AI is needed when movements are not deterministic (e.g., always continue the direction you were following).

There is a stationary, agent-destroying wumpus. Its stench can be sensed immediately to its north, south, east and west. There is an unknown number of “pits”, which emit a breeze similarly. Can we express rules in FOL?

We’ll express the WumpusWorld rules in FOL.

WupusWorld Rules (Tim Finin)

Some Atomic Propositions:

First, we’ll use propositional logic (not FOL).

S12 = There is a stench in cell (1,2)

B34 = There is a breeze in cell (3,4)

W22 = There is a breeze in cell (2,2)

V11 = We have visited cell (1,1)

OK11 = Cell(1,1) is safe.

etc.

Some Rules:

(R1) $\sim{S11}=\gt\, \sim{W11}\land{\sim{W12}}\land{\sim{W21}}$

(R2) $\sim{S21}=\gt\, \sim{W11}\land{\sim{W21}}\land{\sim{W22}}\land{\sim{W31}}$

(R3) $\sim{S11}=\gt\, \sim{W11}\land{\sim{W12}}\land{\sim{W22}}\land{\sim{W13}}$

(R4) $\quad\;{S12}=\gt \quad{W13}\lor\quad{W12}\lor\quad{W22}\lor\quad{W11}$

Rule **R1** means, for example, *If there is no stench in cell (1,1), then the Wumpus is not in cells (1,1), (1,2), or (2,1)*.

Note that the lack of variables requires us to give similar rules for each cell.

Application of FOL to Programming: Dafny

FOL is being applied in to many domains. One is to programming itself—to verify that function code is correct. This is the field of ***program correctness***, whose importance is recognized every time. A company like Boeing loses billions due to incorrect programs.

To carry this out, the preconditions (`requires`) and postconditions (`ensures`) must be precisely specified in FOL, using Dafny syntax.

Automatically Verifying Source Code

```
method MultipleReturns(x: int, y: int)
returns (more: int, less: int)
requires 0 < y
ensures less < x < more
{
      …
}
```

The code example below shows that code can be submitted to Dafny.

Automatically Verifying Source Code (Continue)

```
method MultipleReturns(x: int, y: int)
returns (more: int, less: int)
requires 0 < y
ensures less < x < more
{
      more := x + y;
      less := x - y;
}
```

The following code segment specifies a "find in an array" function. The second `ensures` specifies that a negative value of `index` is returned if `key` is not present in `a`.

Quantifiers … What Does this Express?

```
method Find(a:array<int>, key:int)
returns(index: int)

  ensures 0<=index==>index <a.Length && a[index]==key

  ensures index<0==>forall k::0<=k<a.Length==>a[k]!=key
```

```
        ...
}
```

Dafny is not completely hands-off, it requires an **invariant** for every loop: predicates stating what each iteration of the loop does **not** change. The `while` loop shown in the following code segment has two invariants.

What is this Saying ...?

```
method Find(a:array<int>, key:int)
returns(index: int)

  ensures 0<=index==>index <a.Length && a[index]==key

  ensures index<0==>forall k::0<=k<a.Length==>a[k]!=key

{
  index := 0;
  while index < a.Length
      invariant 0 <= index <= a.Length
      invariant forall k :: 0<=k<index ==> a[k]!=key
   {
      if a[index] == key {return;}
      index := index + 1;
   }
   index := -1;
}
```

Expressing Predicates in Dafny Example

It is possible to name predicates in Dafny (such as `sorted`) and then refer to them (such as in method `Demo`).

```
Is this program correct?
predict sorted(a: array?<int>)
   requires a != null
   reads a
{
   forall j,k::0<=j<k<a.Length==>a[j]<=a[k]
}

method Demo(a:array<int>)
   requires a.Length > 0
            sorted(a)
```

```
ensures a[0] <= a[a.Length-1]

```

Source: https://rise4fun.com/Dafny/SKw6

Inference

Everything Non-imply-able Must be Supplied

Everything in a FOL system that does not follow from a set of existing propositions must be supplied. This can be laborious, as we have seen in Dafny, but the idea is to supply it once and use many times.

Example

One's mother is one's female parent:

$$\forall{m,c}\quad Mother(c)=m\Leftrightarrow{Female(m)}\land{Parent(m,c)}.$$

One's husband is one's male spouse:

$$\forall{w,h}\quad Husband(h,w)\Leftrightarrow{Male(h)}\land{Spouse(h,w)}.$$

Male and female are disjoint categories:

$$\forall{x}\quad Male(x)\Leftrightarrow{\lnot{Female(x)}}.$$

Parent and child are inverse relations:

$$\forall{p,c}\quad Parent(p,c)\Leftrightarrow{Child(c,p)}.$$

A grandparent is a parent of one's parent:

$$\forall{g,c}\quad Grandparent(g,c)\Leftrightarrow{\exists{p}\quad{Parent(g,p)\land{Parent(p,c)}}}.$$

A sibling is another child of one's parents:

$$\forall{x,y}\quad Sibling(x,y)\Leftrightarrow{x\ne{y}\land{\exists{p}}\quad{Parent(p,x)\land{Parent(p,y)}}}.$$

Selected FOL Statements

Loading [Contrib]/a11y/accessibility-menu.js

A FOL system must be provided with a basis for operation. The figure shows selected FOL statements that we would have to provide to WumpusWorld (more later):

The definitions of the **Adjacent** predicate, “`objects can only be at one location at a time,`” and “`a breeze is experiences at a cell if, and only if it is adjacent to a cell containing a pit.`”

$\forall{x,y,a,b}\quad Adjacent([x,y],[a,b])\Leftrightarrow$
$\quad(x=a\land{(y=b-1\lor{y=b+1})})\lor{(y=b\land{(x=a-1\lor{x=a+1})})}$

Applying Quantifiers: Unification

The ***UNIFY quantifier*** in logic creates all solutions to a subset of propositions, within a set of propositions. The result of the first unification is $(\text{trivially}) x = Jane$. The solution to the third is $y = John$ and $x = the\, Mother\, of\, John$. There is no solution to the fourth.

$UNIFY(Knows(John,x),Knows(John,Jane))=\{x/Jane\}$
$UNIFY(Knows(John,x),Knows(y,bill))=\{x/Bill, y/John\}$
$UNIFY(Knows(John,x),Knows(y,Mother(y)))=\{y/John,x/Mother(John)\}$
$UNIFY(Knows(John,x),Knows(x,Elizabeth))=\{fail\}$.

Prolog Logic Programming

The ***Prolog (programming in Logic) language*** was developed decades ago as a programming language based on first-order logic. It has been used, especially in Europe, as an AI language, and even as a general purpose language.

In the example shown, `:-` means “is implied by” or $\Leftarrow$.

The last line is a **query**: asking for solutions to a statement based on a set of FOL statements.

```
man(socrates).
mortal(x) :- man(x).

?- mortal(socrates).
```

Prolog Example

The example shows a set of propositions (using the propositional subset of Prolog).

The second paragfraph shown three queries.

The last paragraph shows what Prolog is capable of doing, even with this small set

```
likes(mary,food).likes(mary,wine).likes(john,wine).
likes(john,mary).
```

The following queries yield the specified answers.

```
| ?- likes(mary,food).yes.
| ?- likes(john,wine).yes.
| ?- likes(john,food).no.
```

```
1. John likes anything that Mary likes
2. John likes anyone who likes wine
3. John likes anyone who likes themselves
```

Source: http://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html

There are many prolog systems, some cloud-based as shown in the figure.

Wumpus World

Here is an interesting nontrivial example: [Hourse Puzzle].

Prolog is good at constraint-base search. For example, it can be used to color maps (i.e., such that adjacent countries are colored differently). Prolog does its best but is potentially inefficient.

The following figure shows a start to the coloring implementation and a reference if you are interested.

Prolog Example: Map Coloring

Example of desired output: `?- colour_countries(Map).`

```
Map = [austria/yellow, belgium/purple,
bulgaria/yellow, croatia/yellow, cyprus/yellow,
czech_republic/purple, denmark/yellow,
estonia/red, finland/yellow, france/yellow,
germany/red, greece/green, hungary/red,
ireland/yellow, italy/red, latvia/green,
luxemburg/green, malta/green, netherlands/yellow,
poland/yellow, portugal/yellow, romania/green,
slovakia/green, slovenia/green, spain/green,
sweden/green, united_kingdom/green]
```

Source:

https://swish.swi-prolog.org/p/Map%20Coloring%20from%20Web.pl

https://www.matchilling.com/introduction-to-logic-programming-with-prolog/

FOL In Program Construction

You can use Dafny to check your FOL alone (i.e., with no programming involved). For example, if you want to verify the that the statements marked \(R\) in the code segment imply the statements marked \(E\), then you can ask, in effect, *if I execute no code, do the postconditions follow from the preconditions?*

dafny Microsoft Research

Is this program correct?

```
method GetMaxIndex(a: array?<int>, k: int, ind: int)
requires /*a[] nontrivial*/ a != null && a.Length > 0;
requires 0 <= ind <= k <= a.Length - 1;
requires forall i :: 0 <= i <= k ==> a[i] <= a[ind];
requires forall j :: 0 <= j < ind ==> a[j] < a[ind];
requires  k >= a.Length - 1;
ensures /*ind an index in a[]*/ 0 <= ind <= a.Length - 1;
ensures /*a[ind] maximal*/ forall i :: 0 <= i <= a.Length - 1 ==> a[i] <= a[ind];
ensures /*ind minimal*/ forall j :: 0 <= j < ind ==> a[j] < a[ind];
{
}
```

R

E

Loading [Contrib]/a11y/accessibility-menu.js

Boston University Metropolitan College

Loading [Contrib]/a11y/accessibility-menu.js