First Order Logic

Formal conceptualization of objects and relations. Prolog is based on predicate calculus; can actually input a program in Prolog just as you would define in logic, and it will search and make inferences for you.

Example objects:

- cs405
- sun
- confucious
- 2 - abstract object
- { integers }
- justice

Universe of Discourse = set of all objects about which knowledge is being expressed

Use predicate calculus to indicate the relations:

Atomic expressions. Predicate(arguments); e.g. ON(a,b).

1. Logical. Linking atomic expressions through conjunction (and), disjunction (or), implication (implies), etc. E.g., ON(a,b) ^ GREEN(a)
2. Quantifiers. Two quantifiers, $\forall$ = Universal, "for all". $\exists$ = Existential, "there exists".

Logic Refresher:

$\wedge$ =AND
$\vee$ =OR

A	$\neg$ A
F	T
T	F

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A → B = ¬A OR B	If A then B (but B may be true independent of A)
F	F	T	
F	T	T	
T	F	F	
T	T	T	

A	B	A ⇔B	A equals B
F	F	T	
F	T	F	
T	F	F	
T	T	T	

Logical Inference Rules

The main rule we will use for inference is Modus Ponens:

$A \rightarrow B$
A
======
B

This is if we know that both $A \rightarrow B$ and A are TRUE, not $((A \rightarrow B) \wedge A) \rightarrow B)$

Another way to view this is to replace $A \rightarrow B$ with $\neg A \vee B$.
Then we get:

$\neg A \vee P$
$A \vee R$
======
$P \vee R$

Write out the truth table of you don't believe it!
We will exploit this rule to prove various hypothesis or propositions.

Ex: Winter $\vee$ Summer
$\neg$Winter $\vee$ Cold
means: Summer $\vee$ Cold

If we try to resolve:

Winter
$\neg$Winter

means: {} empty set. This means a contradiction has arisen, we are stipulating Winter ^ $\neg$Winter at the same time, which is impossible.

Example: Blocks World

U = { A B C D E }

ON(A,B) means block A is on top of block B
Above(A,B) means that A is above B
Clear(X) means nothing is on top of X

The following propositions define our world. Note that these assertions DEFINE the world and must therefore be true all the time. We can't pick and choose for these to be true sometimes and not true other times. Consider them like the laws of physics: they should never be false for ALL possible things in our made-up universe.

$(\forall x \forall y)$ ON(x,y) $\rightarrow$ Above(x,y)
$(\neg \exists y)$ ON(y,x)) $\rightarrow$ Clear(x)
$(\forall x \forall y \forall e)$ ON(x,e) ^ Above(e,y) $\rightarrow$ Above(x,y)

Given rules like these, if we know the configuration of blocks, such as:

ON(A,B).
ON(C,A).

Suppose we want to answer the question, "Is C above A?"
Then we can induce:

ON(A,B) $\rightarrow$ Above(A,B) Map into natural language
ON(C,A) $\rightarrow$ Above(C,A)
On(C,A) ^ Above(A,B) $\rightarrow$ Above(C,B)

Natural language mapping example. The important thing to remember is that these statements are true by definition and dictate what exists in the universe of discourse. We can't make it true for some cases, and false for others as we please.

Some simple sentences:

1. Goober is a good bird.

Bird(Goober) ^ Good(Goober)
GoodBird(Goober)
Good(Goober) ^ ISA(Goober, Bird)

2. If it is not raining tomorrow, Taz will go to Denali.

   $\neg$Weather(rain, tomorrow) $\rightarrow$ Go(Taz, Denali)

3. All cats are mammals.

   $(\forall x)$ Cat(x) $\rightarrow$ Mammal(x)

   Sometimes it is tempting to think that the presence of the condition Cat(x) on the left-hand side of the implication means that somehow the universal quantifier ranges only over cats. This is not technically correct. The universal quantifier makes a statement about EVERYTHING, but it does not make any claim about whether or not non-cats are mammals. If we used the incorrect logic then we could try to express this sentence as:

   $(\forall x)$ Cat(x) ^ Mammal(x)

   But this would be equivalent to saying everything is a cat and a mammal:

   Cat(Kenrick) ^ Mammal(Kenrick) ^
   Cat(Felix) ^ Mammal(Felix) ^
   Cat(Rosebud) ^ Mammal(Rosebud) ^
   …

More complex examples:

U=All living things
Purple(X), Mushroom(X), Poison(X) are our propositions.

1. "All purple mushrooms are poisonous"
$(\forall x)$ Purple(x) ^ Mushroom(x) $\rightarrow$ Poison(x)

This says that for all things X, if X is purple and X is a mushroom, then X is poisonous. Note that this still allows things that are not purple and are not mushrooms to be poisonous, since F $\rightarrow$ Poison(x) is true for all X. When we have an implication, since the entire statement is true if the left half of the implication is false, we can "plug in" values that make the rule true in order to test it.

2. "No purple mushroom is poisonous"
$(\forall x)$ Purple(x) ^ Mushroom(x) $\rightarrow \neg$Poison(x)

Same as above, except for the item is not poisonous.

3. "There is exactly 1 mushroom"
$(\exists x \forall y)$ Mushroom(x) ^ Mushroom(y) ^ x=y

This statement stipulates that there is exactly one mushroom and nothing else in the universe of discourse.

The statement: $(\exists x)$ Mushroom(x) $\rightarrow (\forall y)$ Mushroom(y) $\rightarrow$ x=y

This statement says that for all X, if X is a mushroom then for all Y, if Y is a mushroom then X and Y are the same. This says that there is only one mushroom, but there may also be other things in the universe due to the implication.

4. "There are at least 2 mushrooms"
$(\exists x \exists y)$ Mushroom(x) ^ Mushroom(y) ^ $x \neq y$

This says that there exists some X and some Y such that both X and Y are mushrooms and that X and Y are not the same mushroom. Therefore, there must be at least two mushrooms, and possibly more.

5. "There is at least 1 poisonous mushroom"
$(\exists x)$ (Mushroom(x) ^ Poisonous(x))

This says that there exists some X such that X is a mushroom and it is poisonous. There may be multiple poisonous mushrooms, but there is at least one.

6. "There are at most 2 mushrooms"

This can be somewhat tricky. Here is one attempt:

$(\forall x \forall y \forall z)$ (Mushroom(x) ^ Mushroom(y)) $\rightarrow (\neg (\exists z)$ Mushroom(z) ^ $z \neq x$ ^ $z \neq y$)

For all X and Y such that X and Y are mushrooms, then there does not exist a Z such that Z is also a mushroom and not equal to X or Y. Sounds good, but we want the statement to be true for all X and Y so that it means that there are at most 2 mushrooms. Consider if there are two mushrooms and X and Y are the same mushroom. Then this disallows a second mushroom from existing. Instead we need something like the statement below.

$(\forall x \forall y \forall z)$ (Mushroom(x) ^ Mushroom(y) ^ Mushroom (z)) $\rightarrow$ (z=x $\vee$ z=y $\vee$ x=y)

If you pick any three mushrooms, than at least two of those mushrooms must be the same. Consequently there cannot be more than two mushrooms.

Forward Chaining and Unification

Forward chaining is a method for reasoning about knowledge, and is based on implication. Usually it is used to create inferences when new knowledge is added into the knowledge base, and we want to examine the consequences of that knowledge.

To use forward chaining, we need to convert our clauses into Horn Clauses. This can be done by first converting clauses into CNF using a procedure we'll define later, and then converting the resulting expressions into implications. We will be left with expression of the form $A \vee B$. Use the identity $\neg A \vee B = A \rightarrow B$ to convert into horn clauses.

Typically, new rules are added in Horn Clause format.

$F1(x) \wedge F2(x) \rightarrow F3(x)$
$F4(x) \rightarrow F5(x)$
$F6(x) \wedge F1(x) \wedge F3(x) \rightarrow F7(x)$
…
$LHS \rightarrow RHS$

The forward chaining algorithm entails:

Given new predicate P:
    Add P to KB
    For all rules in the KB, if LHS is true then
        Unify variables (can be tricky to implement)
        Instantiate RHS
        Repeat with RHS

Example if you work for the CIA:
KB Contains:

1) $American(x) \wedge Weapon(y) \wedge Nation(z) \wedge Hostile(z) \wedge Sold(x,z,y) \rightarrow Criminal(x)$
2) $Owns(Krunkonia,x) \wedge Missile(x) \rightarrow Sold(Dr.\ Evil, Krunkonia, x)$
3) $Missile(x) \rightarrow Weapon(x)$
4) $Enemy(x,America) \rightarrow Hostile(x)$

Let's say we know the following:

Dr. Evil is an American.	American(Dr. Evil).
Krunkonia is a nation .	Nation(Krunkonia).
Krunkonia is an enemy of America.	Enemy(Krunkonia, America).
Krunkonia owns object M1	Owns(Krunkonia, M1).
M1 is a missile	Missile(M1).

Now we start adding into our knowledge base using forward chaining:

Forward-Chain(KB, American(Dr. Evil))

5) American(Dr. Evil) Added

This partly completes 1, but not completely. No other matches.

Forward-Chain(KB, Nation(Krunkonia))

6) Nation(Krunkonia) Added

This partly matches 1, but still missing premises so nothing fires.

Forward-Chain(KB, Enemy(Krunkonia,America))

7. Enemy(Krunkonia,America) Added

This matches 4, unify x with Krunkonia and instantiate into Hostile(Krunkonia) then repeat:

Forward-Chain(KB,Hostile(Krunkonia))

8. Hostile(Krunkonia) Added

This matches partly with 1, but not completely, so nothing fires.

Forward-Chain(KB, Owns(Krunkonia,M1)) - Note: M1 is a skolem function

9. Owns(Krunkonia,M1) Added

This partly matches 2, but Missile(x) is missing so nothing fires.

Forward-Chain(KB, Missile(M1))

10. Missile(M1) Added

This matches 2, and we can now infer Sold(Dr. Evil, Krunkonia, M1)

Forward-Chain(KB, Sold(Dr. Evil, Krunkonia, M1)

11. Sold(Dr. Evil, Krunkonia, M1) Added

This almost matches 1, but not quite. Still must exit.

This also matches 3, so unify x with M1 and instantiate into Weapon(M1):

Forward-Chain(KB,Weapon(M1))

12. Weapon(M1) Added

We now finally can satisfy the LHS of rule 1, so we can infer:

13. Criminal(Dr. Evil)

At this point there is nothing left to fire so we are finished.

Notes: The forward chaining process builds up a picture of the situation gradually as new data comes in. Inference not directed towards solving any particular problem. This is called a **data-driven** or **data-directed** procedure since the data drives the operation.

Backward Chaining

Backwards chaining is the opposite of forward chaining. It is typically used to find the answers to a question posed to the knowledge base, or to prove a particular goal.

Given a knowledge base in the form of:

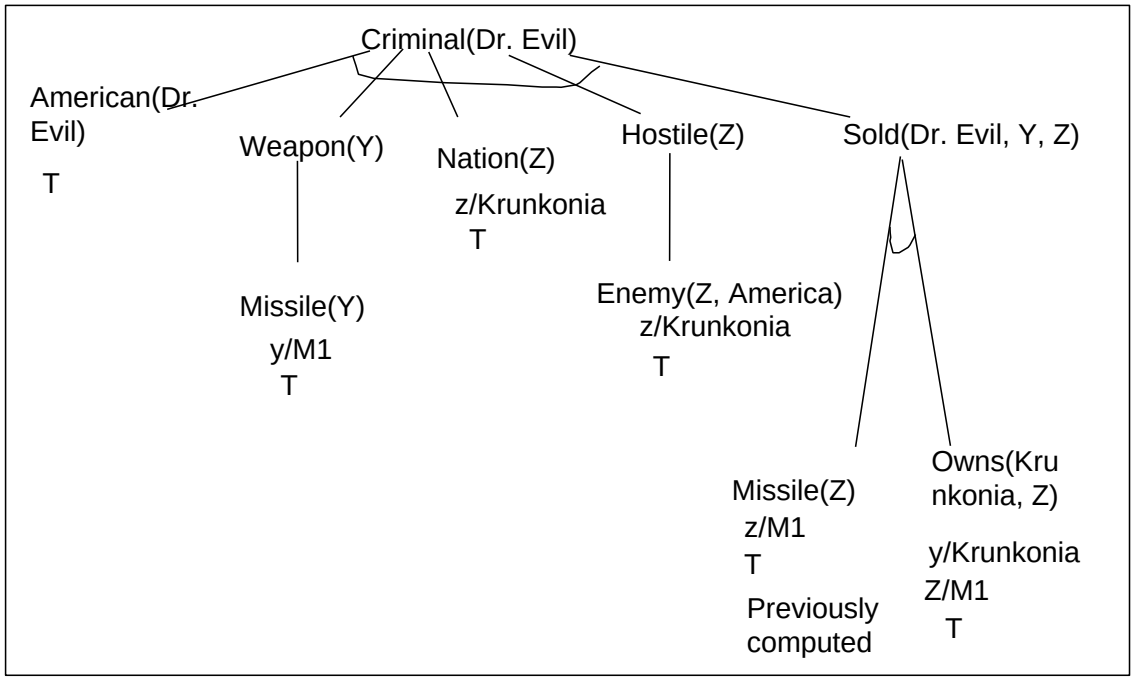
1. F1 $\rightarrow$ F2
2. F3 ^ F5 $\rightarrow$ F4
3. F2 ^ F4 $\rightarrow$ F6
4. F10 … F11 $\rightarrow$ F12

The backwards chaining algorithm entails:

> Given predicate P to prove or ask:
> If P is known to be True in the KB, return true
> Find clause with P on the RHS
> Repeat with every clause on the LHS unifying any variables
> If all clauses true, return true, else return false

Ex: If we want to know if F6 is true, then we need to check and see if F2 and F4 are true. For F2, we need to check and see if F1 is true. For F4, we need to see if F3 and F5 are true. If both are true, we can return and say that F6 is true.

Easy to visualize this graphically:



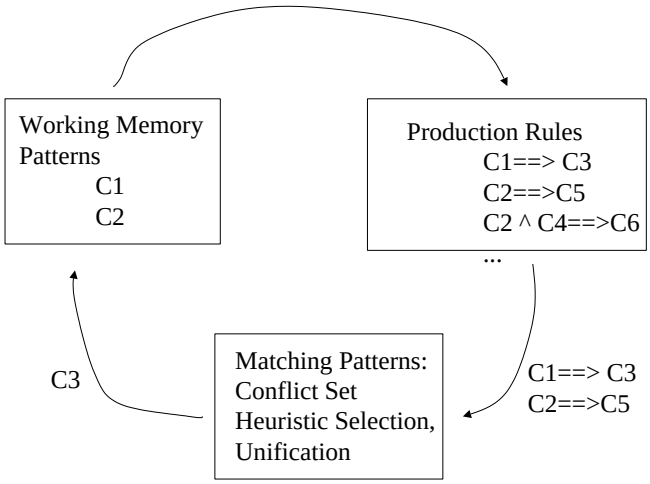
Note that if any of these fail, then the entire chain fails. In the event of a question, all of the predicates that are found may also be returned to the user as rationale behind the reasoning.

Brief overview of **Production Systems:**

The Production Rule system is a model of computation for making inferences and solving problems. We have essentially been using a production rule system all along while performing forward or backwards chaining. However, let's take a step back and define a general framework for applying rules.

There are several components in a production rule system:

1. Production rules. This is a set of rules in the form of "Condition(s) → Action" or "Condition(s)→New Condition".
2. Working Memory. This consists of a description of the current state of the world for the reasoning process. The application of productions can affect this state.
3. Recognize-Act cycle. This cycle begins by applying productions to working memory. The set of productions that match memory is called the *conflict set*. The actions of the conflict set may then fire, and change working memory. The process repeats and terminates when no rules match the working set.
4. Conflict Resolution. This is the process of choosing which rule from the conflict set to fire. This is where heuristic strategies may come into play. Some system employ backtracking if a dead end is encountered.



Many of the commercial expert systems shell engines implement a form of a production rule system.

Logic-Based Application Example : Financial Advisor

As a final example, let's use predicate calculus to represent and reason about a sample problem domain: financial analysis. Although simple, it illustrates many issues involved in real applications.

The function is to advice a user about whether to invest in stocks or savings. Some investors might want to split their money. Let's say that we have determined the following rules:

1. Individuals with inadequate savings should make increasing savings their top priority, regardless of income.
2. Individuals with adequate savings and adequate income should consider stocks, which are riskier but potentially more profit.
3. Individuals with a lower income and already have adequate savings should split their income between savings and stocks.

Our rule is to have at least $5000 in the bank for each dependent. An adequate income must be steady and supply at least $15000 per year plus $4000 for each dependent.

Let's start coding up some rules:

1. Savings_account(inadequate) $\rightarrow$ Investment(savings).
2. Savings_account(adequate) ^ Income(adequate) $\rightarrow$ Investment(stocks).
3. Savings_account(adequate) ^ Income(inadequate) $\rightarrow$ Investment(combination).

Now we have to determine when savings and income are adequate or inadequate. Let's define minsavings and the adequacy of savings:

MinSavings(Z) = 5000*Z

4. $\forall x$ Amount_saved(x) ^ $\exists y$ (dependents(y) ^ greater(x,MinSavings(y))) $\rightarrow$ Savings_Account(adequate)
5. $\forall x$ Amount_saved(x) ^ $\exists y$ (dependents(y) ^ not(greater(x,MinSavings(y)))) $\rightarrow$ Savings_Account(inadequate)

5a. $\neg\exists y$ dependents(y) $\rightarrow$ Savings_Account(adequate)

Now we need to define adequacy of income:

MinIncome(Z) = 15000 + (4000*Z)

6. $\forall x$ Earnings(x, steady) ^ $\exists y$ (dependents(y) ^ greater(x,MinIncome(y))) $\rightarrow$ income(adequate).
7. $\forall x$ Earnings(x, steady) ^ $\exists y$ (dependents(y) ^ Not(greater(x,MinIncome(y)))) $\rightarrow$ income(inadequate).
8. $\forall x$ Earnings(x,unsteady) $\rightarrow$ income(inadequate).

8a. $\forall x$ Earnings(x,steady) ^ $\neg\exists y$ dependents(y) $\rightarrow$ income(adequate)

That's all the knowledge we need for our system. Now let's give our system some sample data for Herman. Toiling as an instructor, Herman makes only $25000 a year. However, he had the good fortune of inheriting some money and has $22000 saved. He has three dependents.

9. amount_saved(22000)
10. earnings(25000, steady)
11. dependents(3)

Let's try to infer an investment strategy for Herman. We can use forward chaining and plug 10 and 11 into the first two components of premise 7.

Earnings(25000, steady) ^ dependents(3)

Unifies with

Earnings(X, steady) ^ dependents(y).

So this means that 25000=X and 3=Y.

Evaluating the function MinIncome(3) yields 15000+12000= 27000. So Not(greater(25000,27000)) is true, and via Modus Ponens we can now infer the RHS of rule 7: 12) income(inadequate).
We can also unify with assertion 4:

Amount_Saved(22000) ^ dependents(3) with 22000=X, 3=Y

Evaluating the function MinSavings(3) yields 15000.

We have Greater(22000,15000) so via Modus Ponens we can now infer the RHS of #4, or 13) savings_account(adequate).

With 12 and 13 we can now fire off rule #3 which suggests Investment(combination).

This is a simple example; more complex rules and knowledge is necessary for really useful advice. You might notice that there is no "magic" in all of this logic; it has essentially been a bunch of if-then-else statements combined with functions. With a large collection of rules, a program can appear magical and mystical but underneath the hood it's still just evaluating a bunch of rules. However some systems may exhibit some "emergent" characteristics, as a collection of rules in combination produce more intelligent behavior than any individual rule.