- Assignment 5 is available
- A solution to Assignment 4 is posted.

What is now required is to give the greatest possible development to mathematical logic, to allow to the full the importance of relations, and then to found upon this secure basis a new philosophical logic, which may hope to borrow some of the exactitude and certainty of its mathematical foundation. If this can be successfully accomplished, there is every reason to hope that the near future will be as great an epoch in pure philosophy as the immediate past has been in the principles of mathematics. Great triumphs inspire great hopes; and pure thought may achieve, within our generation, such results as will place our time, in this respect, on a level with the greatest age of Greece.

– Bertrand Russell, Mysticism and Logic and Other Essays [1917]

Since the midterm...

Done:

- Syntax and semantics of propositional definite clauses
- Model a simple domain using propositional definite clauses
- Bottom-up proof procedure computes a consequence set using modus ponens.
- Top-down proof procedure answers a query using resolution.
- The box model provides a way to procedurally understand the top-down proof procedure with depth-first search.
- Syntax of Datalog: Predicate symbols, constants, variables, queries.
- Semantics of Datalog: Interpretations, variable assignments, models, logical consequence.

Features of Automated Reasoning

- Users can have meanings for symbols in their head.
- The computer doesn't need to know these meanings to derive logical consequence.
- Users can interpret any answers according to their meaning.

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.
An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects and relations in world
  - constants denote individuals
  - predicate symbols denote relations

Formal Semantics

An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- D, the domain, is a nonempty set. Elements of D are individuals.

Formal Semantics

An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- D, the domain, is a nonempty set. Elements of D are individuals.
- ϕ is a mapping that assigns to each constant an element of D. Constant c denotes individual $\phi(c)$.

Formal Semantics

An interpretation is a triple $I = \langle D, \phi, \pi \rangle$, where

- D, the domain, is a nonempty set. Elements of D are individuals.
- ϕ is a mapping that assigns to each constant an element of D. Constant c denotes individual $\phi(c)$.
- π is a mapping that assigns to each n-ary predicate symbol a relation: a function from D^n into $\{true, false\}$.

Clicker Question

Suppose we have an interpretation I with
domain $D = \{$✂, ☎, ✇, ✈, ✎$\}$.
and $\phi(c) =$ ✈ and $\phi(d) =$ ✈.
Which of the following is *not* true:

- A Every statement that is true about d is true about c.
- B c and d refer to two things with the same name
- C There is one individual with two different names
- D c could be replaced by d in all clauses and the same clauses would be true in I

Clicker Question

Given a knowledge base KB, if an answer that is false in the intended interpretation is returned (given a sound and complete proof procedure). Which of the following is true

A One of the clauses in KB must be false in the intended interpretation

B There are too many irrelevant facts that confused the system

C The intended interpretation is a model of KB.

D None of the above.

Clicker Question

Given a knowledge base KB, if g is true in the intended interpretation, and g is not given as an answer (assuming a sound and complete proof procedure that halts), which of the following is true

A One of the clauses in KB must be false in the intended interpretation

B g is false in another model of KB

C The intended interpretation is not a model of KB

D All of the above

E None of the above.

Anonymous variable

- In Prolog, _ is the anonymous variable.
- It is a logical variable where all instances are a different variable.
- _ in queries means we don't care about the value of a variable
- Singleton variables in a clause are often an error. Use _ instead.

Function Symbols

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A classlist.

Function Symbols

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A classlist.
- We extend the notion of term. So that a term can be
  - a variable
  - a constant
  - of form $f(t_1, \ldots, t_n)$ where f is a function symbol and the t_i are terms.

Function Symbols

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A classlist.
- We extend the notion of term. So that a term can be
  - a variable
  - a constant
  - of form $f(t_1, \ldots, t_n)$ where f is a function symbol and the t_i are terms.

Prolog functions are like Haskell constructors (defined with data in Haskell), but don't need to be declared.

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.
An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects and relations in world
  - constants denote individuals (specified by ϕ)
  - predicate symbols denote relations (specified by π)

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.

An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects and relations in world
  - constants denote individuals (specified by ϕ)
  - predicate symbols denote relations (specified by π)
  - in an interpretation and with a variable assignment, term $f(t_1, \ldots, t_n)$ denotes an individual in the domain.

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.
An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects and relations in world
  - constants denote individuals (specified by ϕ)
  - predicate symbols denote relations (specified by π)
  - in an interpretation and with a variable assignment, term $f(t_1, \ldots, t_n)$ denotes an individual in the domain.
    Also specified by ϕ (constant is just a special case).

The semantics is otherwise unchanged.

Semantics: General Idea

A semantics specifies the meaning of sentences in the language.

An interpretation specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects and relations in world
  - constants denote individuals (specified by ϕ)
  - predicate symbols denote relations (specified by π)
  - in an interpretation and with a variable assignment, term $f(t_1, \ldots, t_n)$ denotes an individual in the domain. Also specified by ϕ (constant is just a special case).

The semantics is otherwise unchanged.

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971
- Use $ce(Y, M, D)$ where Y is the year, M is the month and D is the day of the month. (ce is for "common era").

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971
- Use $ce(Y, M, D)$ where Y is the year, M is the month and D is the day of the month. (*ce* is for "common era").
- $ce(\cdot)$ can only be used as part of an atom:
  % $born(Person, Date)$ is true if *Person* was born on *Date*

```
born(justin,ce(1971,dec,25)).
born(pierre,ce(1919,oct,18)).
born(ella_mai,ce(1994,nov,3)).
born(shawn_mendez, ce(1998,aug,8)).
```

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971
- Use $ce(Y, M, D)$ where Y is the year, M is the month and D is the day of the month. (ce is for "common era").
- $ce(\cdot)$ can only be used as part of an atom:
  % $born(Person, Date)$ is true if $Person$ was born on $Date$

```
born(justin,ce(1971,dec,25)).
born(pierre,ce(1919,oct,18)).
born(ella_mai,ce(1994,nov,3)).
born(shawn_mendez, ce(1998,aug,8)).
```

- Define $before(D1, D2)$ which is true when date $D1$ is before date $D2$. (You may use infix predicate $<$ where $X < Y$ is true if X is less than Y).

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971
- Use $ce(Y, M, D)$ where Y is the year, M is the month and D is the day of the month. (ce is for "common era").
- $ce(\cdot)$ can only be used as part of an atom:
  % $born(Person, Date)$ is true if $Person$ was born on $Date$

```
born(justin,ce(1971,dec,25)).
born(pierre,ce(1919,oct,18)).
born(ella_mai,ce(1994,nov,3)).
born(shawn_mendez, ce(1998,aug,8)).
```

- Define $before(D1, D2)$ which is true when date $D1$ is before date $D2$. (You may use infix predicate $<$ where $X < Y$ is true if X is less than Y).
- Add $bce(y, m, d)$ for before common era, e,g,. $bce(55, mar, 15)$ is the ides of March, 55BCE.

Example: dates (dates.pl)

- Suppose we want to refer to dates, e.g., December 25, 1971
- Use $ce(Y, M, D)$ where Y is the year, M is the month and D is the day of the month. (ce is for "common era").
- $ce(\cdot)$ can only be used as part of an atom:
  % $born(Person, Date)$ is true if $Person$ was born on $Date$

```
born(justin,ce(1971,dec,25)).
born(pierre,ce(1919,oct,18)).
born(ella_mai,ce(1994,nov,3)).
born(shawn_mendez, ce(1998,aug,8)).
```

- Define $before(D1, D2)$ which is true when date $D1$ is before date $D2$. (You may use infix predicate $<$ where $X < Y$ is true if X is less than Y).
- Add $bce(y, m, d)$ for before common era, e,g,. $bce(55, mar, 15)$ is the ides of March, 55BCE.
- Given $born(Person, Date)$ information, write $older(P1, P2)$.

Clicker Question

$foo(a, X, X)$

A must be an atom

B must be a term

C Prolog cannot tell if it is a term or atom

D Prolog can tell if it is a term or an atom by where it appears in a clause.

Clicker Question

If $foo(a, X, X)$ appears as

$$foo(a, X, X) \text{ :- } bar(X).$$

For an interpretation I

A $foo(a, X, X)$ denotes an individual in I

B $foo(a, X, X)$ is true or false in I

C $foo(a, X, X)$ denotes an individual in I only given a variable assignment

D $foo(a, X, X)$ is true or false in I only given a variable assignment

Clicker Question

If $foo(a, X, X)$ appears as

$$noon(foo(a, X, X), 17) \text{ :- } bar(X).$$

For an interpretation I

A $foo(a, X, X)$ denotes an individual in I

B $foo(a, X, X)$ is true or false in I

C $foo(a, X, X)$ denotes an individual in I only given a variable assignment

D $foo(a, X, X)$ is true or false in I only given a variable assignment

Working with terms (myis.pl)

- Prolog has a predicate 'is', so that

  $$is(N, E)$$

  usually written

  $$N \text{ is } E$$

  is true when expression E evaluates to number N.
  E must not contain variables when called.

Working with terms (myis.pl)

- Prolog has a predicate 'is', so that

  $$is(N, E)$$

  usually written

  $$N \text{ is } E$$

  is true when expression E evaluates to number N.
  E must not contain variables when called.
- Prolog has predicate $number(N)$ that is true if N is a number.

Working with terms (myis.pl)

- Prolog has a predicate 'is', so that

  $$is(N, E)$$

  usually written

  $$N \text{ is } E$$

  is true when expression E evaluates to number N.
  E must not contain variables when called.
- Prolog has predicate $number(N)$ that is true if N is a number.
- Define $myis(N, E)$ that is true if arithmetic expression E has value the number N.

Working with terms (myis.pl)

- Prolog has a predicate 'is', so that

  $$is(N, E)$$

  usually written

  $$N \text{ is } E$$

  is true when expression E evaluates to number N.
  E must not contain variables when called.
- Prolog has predicate $number(N)$ that is true if N is a number.
- Define $myis(N, E)$ that is true if arithmetic expression E has value the number N.
- 'myis' can be made into an infix operator by declaring:

```
:- op(700, xfx, myis).
```

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.
- The list containing *jan*, *feb* and *mar* is

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.
- The list containing *jan*, *feb* and *mar* is

$$cons(jan, cons(feb, cons(mar, empty)))$$

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.
- The list containing *jan*, *feb* and *mar* is

$$cons(jan, cons(feb, cons(mar, empty)))$$

$member(E, L)$ is true if E is and element of list L.

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.
- The list containing *jan*, *feb* and *mar* is

$$cons(jan, cons(feb, cons(mar, empty)))$$

  $member(E, L)$ is true if E is and element of list L.
- $append(X, Y, Z)$ is true if list Z contains the elements of list X followed by the elements of list Y.

Lists (mylist.pl)

- A list is an ordered sequence of elements.
- Let's use
  - the constant *empty* to denote the empty list, and
  - the function $cons(H, T)$ to denote the list with first element H and rest-of-list T.

  These are not built-in.
- The list containing *jan*, *feb* and *mar* is

$$cons(jan, cons(feb, cons(mar, empty)))$$

  $member(E, L)$ is true if E is and element of list L.
- $append(X, Y, Z)$ is true if list Z contains the elements of list X followed by the elements of list Y.

$$append(empty, Z, Z).$$

$$append(cons(A, X), Y, cons(A, Z)) :- append(X, Y, Z).$$

Syntactic Sugar for Lists (lists.pl)

- The empty list is [ ]
- The list with first element H and the rest of the list T is $[H \mid T]$.

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $member(X, L)$ is true if X is an element of list L

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $member(X, L)$ is true if X is an element of list L
- $append(A, B, C)$ is true if C contains the elements of A followed by the elements of B

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $member(X, L)$ is true if X is an element of list L
- $append(A, B, C)$ is true if C contains the elements of A followed by the elements of B
- $numeq(X, L, N)$ is true if N is the number of instances of X in L.

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $member(X, L)$ is true if X is an element of list L
- $append(A, B, C)$ is true if C contains the elements of A followed by the elements of B
- $numeq(X, L, N)$ is true if N is the number of instances of X in L.
- $sum(L, N)$ is true if N is the sum of the elements of L

Syntactic Sugar for Lists (lists.pl)

- The empty list is $[\,]$
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $member(X, L)$ is true if X is an element of list L
- $append(A, B, C)$ is true if C contains the elements of A followed by the elements of B
- $numeq(X, L, N)$ is true if N is the number of instances of X in L.
- $sum(L, N)$ is true if N is the sum of the elements of L
- $reverse(L, R)$ is true if R is the reverse of list L.