7.7 Agents Based on Propositional Logic

- Already in our extremely simple wumpus-world it turns out that using propositional logic as the knowledge representation language suffers from serious drawbacks.
- To initialize the knowledge base with the rules of the game, we have to give for each square \([x, y]\) the rule about perceiving a breeze (and the corresponding rule for perceiving a stench):

\[
B_{x,y} \Leftrightarrow (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})
\]

- There is at least one wumpus:

\[
W_{1,1} \lor W_{1,2} \lor \cdots \lor W_{4,3} \lor W_{4,4}
\]

and, on the other hand, at most one wumpus, which can be expressed by sentences such as

\[
\neg W_{1,1} \lor \neg W_{1,2}
\]

- When we add rules \(\neg P_{1,1}\) and \(\neg W_{1,1}\) to the KB, the simple \(4 \times 4\) world has required in total \(2 + 2 \cdot 16 + 1 + 120 = 155\) initial sentences containing 64 symbols.
- Model checking should enumerate \(2^{64} \approx 1.8 \times 10^{19}\) possible models.
- More efficient inference algorithms, though, can take advantage of propositional knowledge representation.
- In addition, the KB does not yet contain all the rules of the game (e.g., arrow, gold, walls) and no knowledge of the possible actions of the agent (direction the agent is facing, taking steps), so it cannot yet be used to choose actions.
- In addition to being inefficient, propositional logic is also an unsatisfactory choice.

8 FIRST-ORDER LOGIC

Sentence \(\rightarrow\) AtomicSentence | (Sentence Connective Sentence)
| Quantifier VariableList: Sentence | \(\neg\)Sentence

AtomicSentence \(\rightarrow\) Predicate | Predicate(TermList)

Term \(\rightarrow\) Term | Term, TermList

Term \(\rightarrow\) Function(TermList) | Constant | Variable

Connective \(\rightarrow\) \(\land\) | \(\lor\) | \(\Rightarrow\) | \(\iff\)

Quantifier \(\rightarrow\) \(\exists\) | \(\forall\)

VariableList \(\rightarrow\) Variable | Variable, VariableList

Constant \(\rightarrow\) \(A\) | \(X_i\) | \(\text{John}\) | \(\text{Mary}\)...

Variable \(\rightarrow\) \(a\) | \(x\) | \(s\) | ...

Predicate \(\rightarrow\) \(T\) | \(F\) | \(\text{Before}\) | \(\text{HasColor}\) | \(\text{Raining}\) | ...

Function \(\rightarrow\) \(\text{Mother}\) | \(\text{LeftLeg}\) | ...

- In the realm of first-order predicate logic there are objects, which have properties, and between which there are relations.
- Constant symbols stand for objects.
- Predicate symbols stand for relations, i.e., relations of arity \(n\), \(\text{Mother}(John, Mary)\), and properties are unary relations, \(\text{Student}(John)\).
- A function is a total mapping that associates one single value to the ordered collection of its arguments.
Quantifiers let us express properties of entire collections of objects, instead of enumerating the objects by name. We have the possibility to quantify objects universally and existentially:

\[ \forall y: \text{Student}(y) \Rightarrow \text{EagerToLearn}(y) \]

\[ \exists x: \text{Father}(John) = x \]

To determine the semantics for the syntactic notions (constants, predicates, and functions) need to be bound with an interpretation to correspond (real-)world objects. The world together with the interpretation of the syntax constitutes the model in predicate logic.

Term \( f(t_1, \ldots, t_n) \) refers to the object that is the value of function \( F \) applied to objects \( d_1, \ldots, d_n \), where \( F \) is the interpretation of \( f \) and \( d_1, \ldots, d_n \) are the objects that the argument terms \( t_1, \ldots, t_n \) refer to.

Atomic sentence \( P(t_1, \ldots, t_n) \) is true if the relation referred to by the predicate symbol \( P \) holds among the objects referred to by the arguments \( t_1, \ldots, t_n \).

The semantics of sentences formed with logical connectives is identical to that in propositional logic.

Equality \( = \) is a special relation, whose interpretation cannot be changed. Terms \( t_1 \) and \( t_2 \) have this relation if and only if they refer to the same object.

Because \( \forall \) is really a conjunction over the universe of objects and \( \exists \) is a disjunction, they obey de Morgan’s rules:

\[ \forall x: \neg \varphi(x) \Leftrightarrow \neg \exists x: \varphi(x) \]

\[ \neg \forall x: \varphi(x) \Leftrightarrow \exists x: \neg \varphi(x) \]

\[ \forall x: \varphi(x) \Leftrightarrow \neg \exists x: \neg \varphi(x) \]

\[ \neg \forall x: \neg \varphi(x) \Leftrightarrow \exists x: \varphi(x) \]

The kinship domain:

\[ \forall m, c: \text{Mother}(c) = m \Leftrightarrow \text{Female}(m) \land \text{Parent}(m, c). \]

\[ \forall w, h: \text{Husband}(h, w) \Leftrightarrow \text{Male}(h) \land \text{Spouse}(h, w). \]

\[ \forall x: \text{Male}(x) \Leftrightarrow \neg \text{Female}(x). \]

\[ \forall p, c: \text{Parent}(p, c) \Leftrightarrow \text{Child}(c, p). \]

\[ \forall g, c: \text{Grandparent}(g, c) \Leftrightarrow \exists p: \text{Parent}(g, p) \land \text{Parent}(p, c). \]

\[ \forall x, y: \text{Sibling}(x, y) \Leftrightarrow x \neq y \land \exists p: \text{Parent}(p, x) \land \text{Parent}(p, y). \]
Peano axioms

- Define natural numbers and addition using one constant symbol 0 and a successor function $S$:
  \[ \forall n: \text{NatNum}(n) \Rightarrow \text{NatNum}(S(n)). \]

- So the natural numbers are $0, S(0), S(S(0)), \ldots$

- Axioms to constrain the successor function:
  \[ \forall n: 0 \neq S(n), \forall m, n: m \neq n \Rightarrow S(m) \neq S(n). \]

- Addition in terms of the successor function:
  \[ \forall m: \text{NatNum}(m) \Rightarrow +(m, 0) = m. \]

Axioms for sets

- To define sets we use:
  - Constant symbol $\emptyset$, which refers to the empty set
  - Unary predicate $\exists$, which is true of sets
  - Binary predicates $x \in s$, $s_1 \cup s_2$, $s_1 \cap s_2$, and $\{x|s\}$, which refers to the set resulting from adjoining element $x$ to set $s$

- The only sets are the empty set and those made by adjoining something to a set:
  \[ \forall s: \text{Set}(s) \Leftrightarrow s = \emptyset \lor \exists x, s_2: \text{Set}(s_2) \land s = \{x|s_2\} \]

- The empty set has no elements adjoined into it:
  \[ \neg \exists x, s: \{x|s\} = \emptyset \]

Adjoining an element already in the set has no effect:
\[ \forall x, s: x \in s \Leftrightarrow s = \{x|s\}. \]

The only members of a set are those elements that were adjoined into it:
\[ \forall x, s: x \in s \Leftrightarrow \exists y, s_2: (s = \{y|s_2\} \land (x = y \lor x \in s_2)). \]

Set inclusion:
\[ \forall s_1, s_2: s_1 \subseteq s_2 \Leftrightarrow (\forall x: x \in s_1 \Rightarrow x \in s_2). \]

Two set are equal iff each is a subset of the other:
\[ \forall s_1, s_2: (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \land s_2 \subseteq s_1). \]

\[ \forall x, s_1, s_2: x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \land x \in s_2) \]
\[ \forall x, s_1, s_2: x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \lor x \in s_2). \]

9 INFERENCE IN FIRST-ORDER LOGIC

- By eliminating the quantifiers from the sentences of predicate logic we reduce them to sentences of proposition logic and can turn to familiar inference rules

- We may substitute the variable $\alpha$ in an universally quantified sentence $\forall \alpha: \varphi$ with any ground term, a term without variables

- Substitution $\theta$ is a binding list – set of variable/term pairs – in which each variable is given the ground term replacing it

- Let $\alpha(\theta)$ denote the result of applying the substitution $\theta$ to the sentence $\alpha$

- Universal instantiation to eliminate the quantifier is the inference
  \[ \forall \alpha: \varphi \Rightarrow \alpha(\varphi^g) \]
  for any variable $\alpha$ and ground term $g$
In existential instantiation the variable $v$ is replaced by a Skolem constant $k$ a symbol that does not appear elsewhere in the KB.

$\exists v: a \leftarrow a((v/k))$

E.g., from the sentence $\exists x: \text{Father}(John) = x$ we can infer the instantiation $\text{Father}(John) = F_1$, where $F_1$ is a new constant.

Eliminating existential quantifiers by replacing the variables with Skolem constants and universal quantifiers with the set of all possible instantiations turns the KB essentially propositional.

Ground atomic sentences such as $\text{Student}(John)$ and $\text{Mother}(John, Mary)$ must be viewed as proposition symbols.

Therefore, we can apply any complete propositional inference algorithm to obtain first-order conclusions.

When the KB includes a function symbol, the set of possible ground term substitutions is infinite.

$\text{Father}(\text{Father}(\text{Father}(John)))$

By the famous theorem due to Herbrand (1930) if a sentence is entailed by the original, first-order KB, then there is a proof involving just a finite subset of the propositional knowledge base.

Thus, nested functions can be handled in the order of increasing depth without losing the possibility to prove any entailed sentence.

Inference is hence complete.

Analogy with the halting problem for Turing machines however shows that the problem is undecidable.

More exactly: the problem is semidecidable, the sketched approach comes up with a proof for entailed sentences.

9.2 Unification and Lifting

Inference in propositional logic is obviously too inefficient.

Writing out all variable bindings seems to be futile.

When there is a substitution $\theta$ s.t. $p_i^\theta(\theta) = p_i(\theta)$, for all $i$, where $p_i^\theta$ and $p_i$ are atomic sentences as well as $q$, we can use the Generalized Modus Ponens (GMP):

$p_1^\theta \land \ldots \land p_n^\theta \implies q(\theta)$

For example, from the fact $\text{Student}(John)$ and sentences

$\forall x: \text{EagerToLearn}(x)$ and

$\forall y: \text{Student}(y) \land \text{EagerToLearn}(y) \implies \text{Thesis.2014}(y)$

we can infer $\text{Thesis.2014}(John)$ because of the substitution $\{y/John, x/John\}$.

Generalized Modus Ponens is a sound inference rule.

Similarly as GMP can be lifted from propositional logic to first-order logic, also forward chaining, backward chaining, and the resolution algorithm can be lifted.

A key component of all first-order inference algorithms is unification.

The unification algorithm $\text{Unify}$ takes two sentences and returns a unifier for them if one exists:

$\text{Unify}(p, q) = \theta, \text{ s.t. } p(\theta) = q(\theta)$

otherwise unification fails.
Unify( \(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, Jane)\) ) = \{ x/Jane \}

Unify( \(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Bill})\) ) = \{ x/Bill, y/\text{John} \}

Unify( \(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y))\) ) =
\{ y/\text{John}, x/\text{Mother}(\text{John}) \}

Unify( \(\text{Knows}(\text{John}, x), \text{Knows}(x, \text{Eliza})\) ) = fail

• The last unification fails because \(x\) cannot take on the values \(\text{John}\) and \(\text{Eliza}\) simultaneously
• Because variables are universally quantified, \(\text{Knows}(x, \text{Eliza})\)
  means that everyone knows \(\text{Eliza}\)
• In that sense, we should be able to infer that \(\text{John}\) knows \(\text{Eliza}\)

9.3 Forward Chaining

• As before, let us consider knowledge bases in Horn normal form
• A definite clause either is atomic or is an implication whose body is
  a conjunction of positive literals and whose head is a single positive
  literal

\[ \text{Student}(\text{John}) \]
\[ \text{EagerToLearn}(x) \]
\[ \text{Student}(y) \land \text{EagerToLearn}(y) \Rightarrow \text{Thesis\_2014}(y) \]

• Unlike propositional literals, first-order literals can include variables
• The variables are assumed to be universally quantified

• The problem above arises only because the two sentences happen
to use the same variable name
• The variable names of different sentences have no bearing (prior to
  unification) and one can standardize them apart
• There can be more than one unifier, which of them to return?
• The unification algorithm is required to return the (unique) **most
general unifier**
• The fewer restrictions (bindings to constants) the unifier places, the
  more general it is

\[ \text{Unify}( \(\text{Knows}(\text{John}, x), \text{Knows}(y, z)\) ) \]
\[ \{ y/\text{John}, z/x \} \]
\[ \{ y/\text{John}, x/\text{John}, z/\text{John} \} \]

As in propositional logic we start from facts, and by applying
Generalized Modus Ponens are able to do forward chaining
inference

• One needs to take care that a “new” fact is not just a renaming of a
  known fact

\[ \text{Likes}(x, \text{Candy}) \]
\[ \text{Likes}(y, \text{Candy}) \]

• Since every inference is just an application of Generalized Modus
  Ponens, forward chaining is a sound inference algorithm
• It is also complete in the sense that it answers every query whose
  answers are entailed by any knowledge base of definite clauses
Datalog

- In a Datalog knowledge base the definite clauses contain no function symbols at all
- In this case we can easily prove the completeness of inference
- Let in the knowledge base
  - \( p \) be the number of predicates,
  - \( k \) be the maximum arity of predicates (= the number of arguments), and
  - \( n \) the number of constants
- There can be no more than \( p^k \) distinct ground facts
- So after this many iterations the algorithm must have reached a fixed point, where new inferences are not possible

9.4 Backward Chaining

- In predicate logic backward chaining explores the bodies of those rules whose head unifies with the goal
- Each conjunct in the body recursively becomes a goal
- When the goal unifies with a known fact – a clause with a head but no body – no new (sub)goals are added to the stack and the goal is solved
- Depth-first search algorithm
- The returned substitution is composed from the substitutions needed to solve all intermediate stages (subgoals)
- Inference in Prolog is based on backward chaining

9.4.2 Logic programming

- Prolog, Alain Colmerauer 1972
- Program = a knowledge base expressed as definite clauses
- Queries to the knowledge base
- Closed world assumption: we assume \( \neg \phi \) to be true if sentence \( \phi \) is not entailed by the knowledge base
- Syntax:
  - Capital characters denote variables,
  - Small character stand for constants,
  - The head of the rule precedes the body,
  - Instead of implication use \( :- \),
  - Comma stand for conjunction,
  - Period ends a sentence
- thesis_2014(X):- student(X), eager_to_learn(X).
- Prolog has a lot of syntactic sugar, e.g., for lists and arithmetics
Prolog program \texttt{append(X,Y,Z)} succeeds if list \texttt{Z} is the result of appending (catenating) lists \texttt{X} and \texttt{Y}

\begin{verbatim}
append([],Y,Y).
append([A|X],Y,[A|Z]):- append(X,Y,Z).
\end{verbatim}

Query: \texttt{append([1],[2],Z)?}

\texttt{Z=[1,2]}

We can also ask the query

\texttt{append(A,B,[1,2])?}

Appending what two lists gives the list \texttt{[1,2]}?

As the answer we get back all possible substitutions

\begin{verbatim}
A=[ ] B=[1,2]
A=[1,2] B=[ ]
\end{verbatim}

The clauses in a Prolog program are tried in the order in which they are written in the knowledge base.

Also the conjuncts in the body of the clause are examined in order (from left to right).

There is a set of built-in functions for arithmetic, which need not be inferred further.

- E.g., \texttt{X is 4+3 \rightarrow X=7}

For instance I/O is taken care of using built-in predicates that have side effect when executed.

Negation as failure

\begin{verbatim}
alive(X):- not dead(X).
\end{verbatim}

"Everybody is alive if not provably dead"

The negation in Prolog does not correspond to the negation of logic (using the closed world assumption)

\begin{verbatim}
single_student(X):-
    not married(X), student(X).
student(peter).
marrried(john).
\end{verbatim}

By the closed world assumption, \texttt{X=peter} is a solution to the program.

The execution of the program, however, fails because when \texttt{X=john} the first predicate of the body fails.

If the conjuncts in the body were inverted, it would succeed.

An equality goal succeeds if the two terms are unifiable.

- E.g., \texttt{X+Y=2+3 \rightarrow X=2, Y=3}

Prolog omits some necessary checks in connection of variable bindings \(
\rightarrow \) Inference is not sound.

These are seldom a problem.

Depth-first search can lead to infinite loops (= incomplete)

\begin{verbatim}
path(X,Z):- path(X,Y), link(Y,Z).
path(X,Z):- link(X,Z).
\end{verbatim}

Careful programming, however, lets us escape such problems.

\begin{verbatim}
path(X,Z):- link(X,Z).
path(X,Z):- path(X,Y), link(Y,Z).
\end{verbatim}
Anonymous variable  
member(X,[X|_]).
member(X,[_|Y]):- member(X,Y).

Works just fine 
member(d,[a,b,c,d,e,f,g])?yes
member(2,[3,a,4,f])?no

But queries  
member(a,X)?member(a,[a,b,r,a,c,a,d,a,b,r,a])?  
do not necessarily give the intended answers

We can explicitly prune the execution of Prolog programs by cutting  
Negation using cut  
not X:- X,! fail.
not X.

fail causes the program to fail  
At the point of a cut all bindings that have been made since starting to examine the rule are fixed  
For the conjuncts in the body preceding the cut, no new solutions are searched for  
Neither does one examine other rules having the same head

Prolog may come up with the same answer through several inference paths  
Then the same answer is returned more than once  
minimum(X,Y,X):- X<=Y.
minimum(X,Y,Y):- X>=Y.

Both rules yield the same answer for the query minimum(2,2,M)?  
One must be careful in using cut for optimizing inference  
minimum(X,Y,X):- X<=Y, !.
minimum(X,Y,Y).

This program is erroneous, for instance minimum(2,8,8) holds according to it

The key question of Prolog and logic programming obviously is efficiency of execution  
Prolog implementations use a wide variety of enhancement techniques  
For example, instead of generating all possible solutions for a subgoal before examining the next subgoal, a Prolog interpreter is content (so far) with just one  
Similarly variable binding is at each instant unique; only when the search runs into a dead end, can backing up to a choice point lead to unbinding of variables  
A stack of history, called the trail, needs to be maintained to keep track of all variable bindings