Logic Programming

Theory Lecture 3: Definite Clause Predicate Logic

Alex Simpson

School of Informatics

7th October 2013

Predicate logic / predicate calculus / first-order logic

So far, we have looked only at *propositional logic*, where formulas are built from *propositional atoms* which express (indecomposable) *propositions* (statements which are either true or false).

Predicate logic has a richer vocabulary and expressivity.

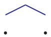
- Its *terms* represent elements in an assumed world of discourse called the *universe*
- Its *predicates* express relationships between these elements.
- Its *formulas* express propositions (statements that are either true of false) about the universe.

Example universe and terms

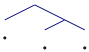
Unlabelled binary trees.

E.g.

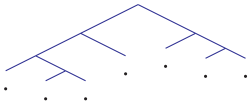
`leaf`



`node(leaf,leaf)`



`node(leaf,node(leaf,leaf))`



`node(node(node(leaf,node(leaf,leaf)),leaf),node(leaf,node(leaf,leaf)))`

Example predicates

`reflection/2`

`reflection(node(node(leaf,leaf),leaf), node(leaf,node(leaf,leaf)))`
`¬reflection(node(node(leaf,leaf),leaf), node(node(leaf,leaf),leaf))`

`symmetric/1`

`symmetric(node(node(node(leaf,leaf),leaf),node(leaf,node(leaf,leaf))))`
`¬symmetric(node(node(node(leaf,leaf),leaf),node(node(leaf,leaf),leaf)))`

Example logic program and query

For predicate logic (just as for propositional logic), a query is a formula and a program is a collection of formulas (the knowledge base).

Program:

$$\texttt{reflection(leaf,leaf)}$$

$$\texttt{reflection(S1,T1)} \wedge \texttt{reflection(S2,T2)} \longrightarrow \texttt{reflection(node(S1,S2),node(T2,T1))}$$

$$\texttt{reflection(T,T)} \longrightarrow \texttt{symmetric(T)}$$

Query:

$$\texttt{symmetric(X)}$$

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
| ?- symmetric(X).
```

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
 | ?- symmetric(X).
 X = leaf
```

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
 | ?- symmetric(X).
 X = leaf
 X = node(leaf,leaf)
```

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
 | ?- symmetric(X).
 X = leaf
 X = node(leaf,leaf)
 X = node(node(leaf,leaf),node(leaf,leaf))
```

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
 | ?- symmetric(X).
 X = leaf
 X = node(leaf,leaf)
 X = node(node(leaf,leaf),node(leaf,leaf))
 X = node(node(leaf,node(leaf,leaf)),node(node(leaf,leaf),leaf))
```

Let's try this in Sicstus Prolog

Program:

```
reflection(leaf,leaf).
reflection(node(S1,S2),node(T2,T1)) :- reflection(S1,T1), reflection(S2,T2).
symmetric(T) :- reflection(T,T).
```

Query:

```
| ?- symmetric(X).
X = leaf
X = node(leaf,leaf)
X = node(node(leaf,leaf),node(leaf,leaf))
X = node(node(leaf,node(leaf,leaf)),node(node(leaf,leaf),leaf))
X = ...
```

Issues to address

- Why are these answers correct?
- How Prolog computes the answers
- Prolog does not find all correct answers, though it would be possible for it to do so in principle

Issues to address

- Why are these answers correct?
  (Logical consequence — today's lecture)

- How Prolog computes the answers

- Prolog does not find all correct answers, though it would be possible for it to do so in principle

Issues to address

- Why are these answers correct?
  (Logical consequence — today's lecture)
- How Prolog computes the answers
  (Proof search — Theory Lecture 4)
- Prolog does not find all correct answers, though it would be possible for it to do so in principle

Issues to address

- Why are these answers correct?
  (Logical consequence — today's lecture)
- How Prolog computes the answers
  (Proof search — Theory Lecture 4)
- Prolog does not find all correct answers, though it would be possible for it to do so in principle
  (Incompleteness and completeness — Theory Lecture 5)

Issues to address

- Why are these answers correct?
  (Logical consequence — today's lecture)
- How Prolog computes the answers
  (Proof search — Theory Lecture 4)
- Prolog does not find all correct answers, though it would be possible for it to do so in principle
  (Incompleteness and completeness — Theory Lecture 5)

The story is very similar to that of Theory Lectures 1–2, except that we are now considering the richer paradigm of *predicate logic*, rather than just propositional logic.

Predicate logic — terms

Terms are built from *variables*, *constants* and *function symbols*.

Grammar of terms:

$$
\begin{aligned}
\textit{term} \ &::= \textit{var} \\
&\quad \mid \textit{constant} \\
&\quad \mid \textit{fn\_symbol}\,(\textit{term\_list}) \\[1em]
\textit{term\_list} \ &::= \textit{term} \\
&\quad \mid \textit{term}, \textit{term\_list}
\end{aligned}
$$

In example:

constants:	`leaf`
function symbols:	`node/2`
variables:	`S1,S2,T1,T2,T,X`

Predicate logic — formulas

Formulas are built from *atomic formulas* using *connectives* $\neg, \wedge, \vee, \longrightarrow$ and *quantifiers* $\forall, \exists$.

Grammar of formulas:

$$
\begin{aligned}
\mathit{form} \;::=\;& \mathit{predicate}\,(\mathit{term\_list}) && \text{(atomic formula)}\\
\mid\;& \neg \mathit{form} \\
\mid\;& \mathit{form} \wedge \mathit{form} \\
\mid\;& \mathit{form} \vee \mathit{form} \\
\mid\;& \mathit{form} \longrightarrow \mathit{form} \\
\mid\;& \forall \mathit{var}.\ \mathit{form} \\
\mid\;& \exists \mathit{var}.\ \mathit{form}
\end{aligned}
$$

In example:

predicate symbols: `reflection/2`, `symmetric/1`

Note on syntax of terms and formulas

Notice how formulas in predicate logic are carefully structured:

- *Terms:* built from variables, constants and function symbols.
- *Atomic formulas:* single predicate symbol with list of terms.
- *Formulas:* built from atomic formulas using connectives and quantifiers.

So, for example,

```
reflection(node(leaf,X),Y)
```

is a legitimate atomic formula, but

```
node(reflection(leaf,X),Y)
```

is ill-formed because a predicate symbol (`reflection`) appears inside a function symbol (`node`).
In contrast, in Prolog, there is no syntactic restriction on how operators can be applied. Nevertheless, we shall assume that the rules of Predicate-logic syntax are followed. (This is advisable in practice since it aids the understandability of code.)

Motivating structures

Recall from slide 2:

- *Terms* represent elements in an assumed world of discourse called the *universe*
- *Predicates* express relationships between these elements.
- *Formulas* express propositions (statements that are either true of false) about the universe.

For example, the formula

$$\exists\, \mathtt{T}.\, \mathtt{symmetric}(\mathtt{T})$$

says:

> there exists an element $\mathtt{T}$ of the universe such that the property $\mathtt{symmetric}$ holds of $\mathtt{T}$.

Whether this is true or not depends upon the choice of universe and on how we specify the interpretation of constants, function symbols and predicates in the universe.

Motivating structures

Recall from slide 2:

- *Terms* represent elements in an assumed world of discourse called the *universe*
- *Predicates* express relationships between these elements.
- *Formulas* express propositions (statements that are either true of false) about the universe.

For example, the formula

$$\exists\, \mathtt{T}.\, \mathtt{symmetric}(\mathtt{T})$$

says:

> there exists an element `T` of the universe such that the property `symmetric` holds of `T`.

Whether this is true or not depends upon the choice of universe and on how we specify the interpretation of constants, function symbols and predicates in the universe.

This information is provided by the notion of *structure*.

"I don't know what you mean by 'glory'," Alice said.

Humpty Dumpty smiled contemptuously. "Of course you don't—till I tell you. I meant 'there's a nice knock-down argument for you!'"

"But 'glory' doesn't mean 'a nice knock-down argument'," Alice objected.

"When I use a word," Humpty Dumpty said, in rather a scornful tone, "it means just what I choose it to mean—neither more nor less."

Lewis Carroll, *Through the Looking Glass*, Ch. VI

Structures

A *structure* $\mathcal{S}$ is given by:

- A set U, called the *universe*.
- For each constant $\mathtt{c}$, an associated element $\mathtt{c}^{\mathcal{S}}$ of the universe.
- For each function symbol $\mathtt{f}/n$, an associated n-argument function $\mathtt{f}^{\mathcal{S}} \colon U^n \to U$.
- For each predicate symbol $\mathtt{p}/n$, an associated n-argument function $\mathtt{p}^{\mathcal{S}} \colon U^n \to \{\mathbf{true}, \mathbf{false}\}$.

(N.B., we write U^n for the set of n-tuples of elements of U.)

The notion of structure plays a role for predicate logic analogous to that played by *interpretation* for propositional logic.

Example structure $\mathcal{S}_1$

The *"intended model"* of our example language

- U is the set of unlabelled binary trees.
- $\texttt{leaf}^{\mathcal{S}_1}$ is the leaf tree $\cdot$.
- $\texttt{node}^{\mathcal{S}_1}$ is the function:

$$(T_1, T_2) \;\mapsto\; \begin{array}{c} \diagup\!\diagdown \\ T_1 \quad T_2 \end{array}$$

- $\texttt{reflection}^{\mathcal{S}_1}$ and $\texttt{symmetric}^{\mathcal{S}_1}$ are the expected relations

$$\begin{aligned} \texttt{reflection}(T_1, T_2) = \textbf{true} \quad &\Leftrightarrow \quad T_1 \text{ is the reflection of } T_2 \\ \texttt{symmetric}(T) = \textbf{true} \quad &\Leftrightarrow \quad T \text{ is symmetric} \end{aligned}$$

illustrated on "Example predicates" slide.

Example structure $\mathcal{S}_2$

We interpret the language over a different universe.

- U is the set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ of natural numbers
- $\texttt{leaf}^{\mathcal{S}_2}$ is the number 0
- $\texttt{node}^{\mathcal{S}_2}$ is the function:

$$(n_1, n_2) \;\mapsto\; \max(n_1, n_2) + 1$$

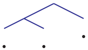
- $\texttt{reflection}^{\mathcal{S}_2}$ and $\texttt{symmetric}^{\mathcal{S}_2}$ are defined by:

$$\begin{aligned} \texttt{reflection}^{\mathcal{S}_2}(n_1, n_2) = \textbf{true} \quad &\Leftrightarrow \quad n_1 = n_2 \\ \texttt{symmetric}^{\mathcal{S}_2}(n) \quad &= \quad \textbf{true} \end{aligned}$$

Example structure $\mathcal{S}_3$

A more aribtrarily chosen structure.

- U is the set of unlabelled binary trees.
- $\texttt{leaf}^{\mathcal{S}_3}$ is the tree

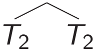


- $\texttt{node}^{\mathcal{S}_3}$ is the function:

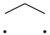
$$(T_1, T_2) \mapsto T_1$$

- $\texttt{reflection}^{\mathcal{S}_3}$ and $\texttt{symmetric}^{\mathcal{S}_3}$ are defined by:

$$\texttt{reflection}^{\mathcal{S}_3}(T_1, T_2) = \textbf{true} \quad \Leftrightarrow \quad T_1 =$$



$$\texttt{symmetric}^{\mathcal{S}_3}(T) = \textbf{true} \quad \Leftrightarrow \quad T =$$



Interpretation of terms in a structure $\mathcal{S}$

A *variable assignment* is a function ρ mapping variables to elements in the universe U.

So, for every variable $\mathtt{X}$, we have an associated element $\rho(X) \in U$.

The function ρ is extended from variables to all terms by:

$$\begin{array}{rcll} \rho(\mathtt{c}) & = & \mathtt{c}^{\mathcal{S}} & \mathtt{c} \text{ a constant} \\ \rho(\mathtt{f}(t_1, \ldots, t_n)) & = & \mathtt{f}^{\mathcal{S}}(\rho(t_1) \ldots, \rho(t_n)) & \mathtt{f}/n \text{ a function symbol} \end{array}$$

So, for every term t, we have an associated element $\rho(t) \in U$.

Satisfaction of a formula F in a structure $\mathcal{S}$

Let $\mathcal{S}$ be a structure and ρ a variable assignment.

The next slide defines the *satisfaction* relation

$$\mathcal{S} \models_\rho F$$

This means:

> F *is true in* $\mathcal{S}$ *(under variable assignment* ρ*).*

A formula is said to be *closed* (or a *sentence*) if every variable $\mathtt{X}$ in the formula occurs inside the scope of a quantifier $\forall\,\mathtt{X}.$ or $\exists\,\mathtt{X}.$ (the quantifier is said to *bind* the variable).

If a formula F is closed then the relationship $\mathcal{S} \models_\rho F$ is independent of ρ, so we can just write $\mathcal{S} \models F$

Definition of satisfaction relation

$$
\begin{array}{lcl}
\mathcal{S} \models_\rho \mathtt{p}(t_1,\ldots,t_n) & \Leftrightarrow & \mathtt{p}^{\mathcal{S}}(\rho(t_1),\ldots,\rho(t_n)) = \textbf{true} \\
\mathcal{S} \models_\rho \neg F & \Leftrightarrow & \text{it is not the case that } \mathcal{S} \models_\rho F \\
\mathcal{S} \models_\rho F_1 \wedge F_2 & \Leftrightarrow & \mathcal{S} \models_\rho F_1 \text{ and } \mathcal{S} \models_\rho F_2 \\
\mathcal{S} \models_\rho F_1 \vee F_2 & \Leftrightarrow & \mathcal{S} \models_\rho F_1 \text{ or } \mathcal{S} \models_\rho F_2 \\
\mathcal{S} \models_\rho F_1 \rightarrow F_2 & \Leftrightarrow & \mathcal{S} \models_\rho F_1 \text{ implies } \mathcal{S} \models_\rho F_2 \\
\mathcal{S} \models_\rho \forall \mathtt{X}.\, F & \Leftrightarrow & \text{for all } a \in U\text{, we have } \mathcal{S} \models_{\rho[\mathtt{X}:=a]} F \\
\mathcal{S} \models_\rho \exists \mathtt{X}.\, F & \Leftrightarrow & \text{there exists } a \in U \text{ s.t. } \mathcal{S} \models_{\rho[\mathtt{X}:=a]} F
\end{array}
$$

Here, $\rho[\mathtt{X} := a]$ is the *modified variable assignment* defined by:

$$
\begin{array}{lcll}
\rho[\mathtt{X} := a]\,(\mathtt{X}) & = & a & \\
\rho[\mathtt{X} := a]\,(\mathtt{Y}) & = & \rho(\mathtt{Y}) & \qquad \mathtt{Y} \text{ any variable other than } \mathtt{X}
\end{array}
$$

Logical consequence

A formula G is said to be a *logical consequence* of formulas $F_1, F_2, \ldots, F_n$, notation

$$F_1, \ldots, F_n \models G \quad ,$$

iff, for all structures $\mathcal{S}$ and all variable assignments ρ,

$$\text{if } \mathcal{S} \models_\rho F_1 \text{ and } \ldots \text{ and } \mathcal{S} \models_\rho F_n \text{ then } \mathcal{S} \models_\rho G \quad .$$

In the case that $F_1, F_2, \ldots, F_n, G$ are sentences (i.e., closed formulas), we can simplify this to: for all structures $\mathcal{S}$,

$$\text{if } \mathcal{S} \models F_1 \text{ and } \ldots \text{ and } \mathcal{S} \models F_n \text{ then } \mathcal{S} \models G \quad .$$

Model

A structure $\mathcal{S}$ is said to be a *model* of the sentences $F_1, \ldots, F_n$ if

$$\mathcal{S} \models F_1 \quad \textit{and} \quad \ldots \quad \textit{and} \quad \mathcal{S} \models F_n \ .$$

We can rephrase logical consequence using the notion of model.

For all sentences $F_1, \ldots, F_n, H$, the logical consequence

$$F_1, \ldots, F_n \models H$$

holds if and only if,

$$\forall \text{ models } \mathcal{S} \text{ of } F_1, \ldots, F_n, \quad \mathcal{S} \models H \ .$$

Examples

Consider our example program (universally quantified)

$$\texttt{reflection(leaf,leaf)}$$

$$\forall\, \texttt{S1,S2,T1,T2}.\ \texttt{reflection(S1,T1)} \land \texttt{reflection(S2,T2)} \longrightarrow \texttt{reflection(node(S1,S2),node(T2,T1))}$$

$$\forall\, \texttt{T}.\ \texttt{reflection(T,T)} \longrightarrow \texttt{symmetric(T)}$$

Examples

Consider our example program (universally quantified)

$$
\begin{array}{rl}
 & \texttt{reflection(leaf,leaf)} \\
\forall\, \texttt{S1,S2,T1,T2}.\ \texttt{reflection(S1,T1)} & \wedge\, \texttt{reflection(S2,T2)} \\
 & \rightarrow \texttt{reflection(node(S1,S2),node(T2,T1))} \\
\forall\, \texttt{T}.\ \texttt{reflection(T,T)} & \rightarrow \texttt{symmetric(T)}
\end{array}
$$

- Structure $\mathcal{S}_1$ *is* a model of our example program.

Examples

Consider our example program (universally quantified)

$$\texttt{reflection(leaf,leaf)}$$
$$\forall \texttt{S1,S2,T1,T2}.\ \texttt{reflection(S1,T1)} \land \texttt{reflection(S2,T2)} \rightarrow \texttt{reflection(node(S1,S2),node(T2,T1))}$$
$$\forall \texttt{T}.\ \texttt{reflection(T,T)} \rightarrow \texttt{symmetric(T)}$$

- Structure $\mathcal{S}_1$ *is* a model of our example program.
- Structure $\mathcal{S}_2$ *is* a model of our example program too.

Examples

Consider our example program (universally quantified)

$$\begin{array}{rl} & \texttt{reflection(leaf,leaf)} \\ \forall\,\texttt{S1,S2,T1,T2}.\ \texttt{reflection(S1,T1)} & \land\ \texttt{reflection(S2,T2)} \\ & \rightarrow \texttt{reflection(node(S1,S2),node(T2,T1))} \\ \forall\,\texttt{T}.\ \texttt{reflection(T,T)} & \rightarrow \texttt{symmetric(T)} \end{array}$$

- Structure $\mathcal{S}_1$ *is* a model of our example program.
- Structure $\mathcal{S}_2$ *is* a model of our example program too.
- Structure $\mathcal{S}_3$ *is not* a model of our example program because, for example, `reflection(leaf,leaf)` does not hold.

Predicate logic is too expressive for computation

In propositional logic, logical consequence is *decidable*, albeit inefficiently.

In predicate logic, logical consequence is not decidable. However it is *semidecidable*: there exists a complete proof search procedure that is guaranteed to find a proof of a logical consequence when the consequence holds, but never terminates when the consequence doesn't hold.

Such general proof search is too inefficient to constitute a means of computation. (For example, a search to see if the Riemann Hypothesis is a consequence of Zermelo-Fraenkel Set Theory is unlikely to terminate before the million dollar prize has become worthless due to inflation.) So general predicate logic is unsuitable for a logic programming language.

As in the propositional case, we restrict to *definite clause logic*.

Definite clauses in predicate logic

A *definite clause* is a formula of one of the two shapes below

$$B \qquad \text{(a } \textit{fact}\text{)}$$

$$A_1 \wedge \cdots \wedge A_k \;\rightarrow B \qquad \text{(a } \textit{rule}\text{)}$$

where $A_1, \ldots, A_k, B$ are all *atomic formulas*, that is, formulas of the simple form $\mathtt{p}(t_1, \ldots, t_n)$ where $\mathtt{p}$ is a predicate symbol.

A *logic program* is a list $F_1, \ldots, F_n$ of definite clauses

The clauses in the program $F_1, \ldots, F_n$ are understood as implicitly as *universally quantified* closed formulas

$$\forall \mathit{Vars}(F_1).\, F_1, \;\ldots\; , \forall \mathit{Vars}(F_n).\, F_n$$

Goals in definite clause logic

A *goal* is a list $G_1, \ldots, G_m$ of atomic formulas.

The job of the system is to ascertain whether the logical consequence below holds.

$$\forall \mathit{Vars}(F_1).\, F_1, \ldots, \forall \mathit{Vars}(F_n).\, F_n \models \exists \mathit{Vars}(G_1, \ldots, G_m).\, G_1 \wedge \cdots \wedge G_m \quad .$$

The atomic formulas in the query $G_1, \ldots, G_n$ are thus understood as implicitly *existentially quantified*

Example: The goal list `reflection(S,T), symmetric(T)` is understood as the existentially quantified closed formula

$$\exists\, \texttt{S,T}.\ \texttt{reflection(S,T)} \wedge \texttt{symmetric(T)}$$

In fact the system does more that ascertain that

$$\exists Vars(G_1, \ldots, G_m).\ G_1 \wedge \cdots \wedge G_m$$

is a logical consequence ot the theory.

The system finds a *substitution* (of terms for variables) which supplies *witnesses* for the existentially quantified variables..

This is once again achieved by a top-down proof search procedure, which will be the topic of the next lecture.

Examples

It is a logical consequence of our example program that:

$$\exists\, \mathtt{T}.\, \mathtt{symmetric}(\mathtt{T})$$

In particular, it is a consequence that `symmetric(leaf)`.

However, it is not a logical consequence that

$$\exists\, \mathtt{T}.\, \neg\mathtt{symmetric}(\mathtt{T})$$

Because, $\mathcal{S}_2$ is a model of the program and:

$$\mathcal{S}_2 \models \forall\, \mathtt{T}.\, \mathtt{symmetric}(\mathtt{T})$$

So not every sentence that is true in our "intended model" $\mathcal{S}_1$ is a logical consequence of our theory.

Prospectus

Because of the use of negation, $\exists\, \texttt{T}.\, \neg\texttt{symmetric(T)}$ is not a legitimate query in definite-clause logic.

For a definite clause goal $G_1, \ldots, G_m$ (in our example language for trees) it is the case that $\exists \mathit{Vars}(G_1, \ldots, G_m).\, G_1 \wedge \cdots \wedge G_m$ is a logical consequence of the example program if and only if it is true in the intended model $\mathcal{S}_1$.

In general, we shall see (Lecture 6) that every definite clause theory has an "intended model", its minimum Herbrand model, and that a definite-clause query is a logical consequence of the theory if and only if it is true in this model.

Main points today

predicate logic terms and formulas

structures and the satisfaction relation

logical consequence for predicate logic

notion of model

definite clauses, programs and goals