INST0072 Lecture 9: Logic, Prolog, and Clark Completion

1. In the Last Lecture

In the last lecture we saw that

- A Prolog program can include *meta-level* clauses about itself.
- Three commonly used predefined meta-level predicates are ‘clause/2’, ‘assertz/1’ and ‘retract/1’.
- The 'clause/2' predicate can be used in *meta-interpreters* that refine or adapt Prolog's search strategy for specific problems.
- The meaning of Prolog programs in terms of classial logic is not always straightforward, partly because Prolog includes *Uniquness-of-names* and *Closed World Assumptions*.

[index]

2. Prolog and Resolution

For a Prolog program in which all the predicates have arity 0 (i.e. no arguments) and that does not use negation-by-failure, a successful branch in the search tree of a query is equivalent to a propositional calculus proof by contradiction using resolution.

Reminder: the general resolution inference rule is:

$$\frac{(L_1 \vee \ldots \vee L_i \vee p \vee L_{i+1} \vee \ldots \vee L_m), \quad (L'_1 \vee \ldots \vee L'_j \vee \neg p \vee L'_{j+1} \vee \ldots \vee L'_n)}{(L_1 \vee \ldots \vee L_m \vee L'_1 \vee \ldots \vee L'_n)}$$

For example:

$$\frac{(big \vee soft \vee \neg red), \quad (\neg soft \vee \neg new)}{(big \vee \neg red \vee \neg new)} \qquad \frac{(soft), \quad (\neg soft)}{\perp}$$

[index]

3. Reminder: Resolution Proof By Contradiction

The general theorem (from Lecture 3, Slide 12):

Resolution Calculus Soundness and Completeness

For any CNF knowledge base KB and formula F

$$KB \models F \quad \text{if and only if} \quad KB \cup \mathbf{cnfset}[\neg F] \vdash_{res} \bot$$

... and a special (simpler) case if F is a single proposition p:

For any CNF knowledge base KB and proposition p

$$KB \models p \quad \text{if and only if} \quad KB \cup \{\neg p\} \vdash_{res} \bot$$

For example: $\{(\neg raining \vee wet), (raining)\} \models wet$ if and only if $\{(\neg raining \vee wet), (raining), (\neg wet)\} \vdash_{res} \bot$

[index]

4. Example Resolution Proof By Contradiction

- Finish the following proof of

$$\{(\neg raining \vee wet), (raining), (\neg wet)\} \vdash_{res} \bot$$

by adding the final two steps (4) and (5):

(1)	$(\neg raining \vee wet)$	by assumption
(2)	$(raining)$	by assumption
(3)	(wet)	by (1), (2), resolution

4. Example Resolution Proof By Contradiction

- Finish the following proof of

$$\{(\neg raining \vee wet), (raining), (\neg wet)\} \vdash_{res} \bot$$

by adding the final two steps (4) and (5):

(1)	$(\neg raining \vee wet)$	by assumption
(2)	$(raining)$	by assumption
(3)	(wet)	by (1), (2), resolution
(4)	$(\neg wet)$	by assumption
(5)	$\bot$	by (3), (4), resolution

so by resolution soundness and completeness this shows that:

$$\{(\neg raining \vee wet), (raining)\} \models wet$$

5. An Example Prolog Proof

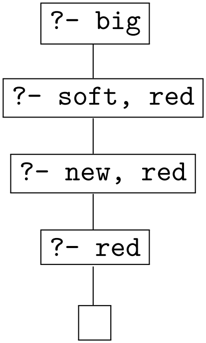
Prolog program:

```
big :- soft, red.
soft :- new.
new.
red.
```

Query:

```
?- big.
```

Search tree:



[index]

6. Transforming Programs Into CNF Theories

Prolog program:	In logic:	In CNF:
`big :- soft, red.`	$big \leftarrow (soft \wedge red)$	$(big \vee \neg soft \vee \neg red)$
`soft :- new.`	$soft \leftarrow new$	$(soft \vee \neg new)$
`new.`	new	(new)
`red.`	red	(red)

$$
\begin{aligned}
& big \leftarrow (soft \wedge red) & \\
\equiv\ & big \vee \neg(soft \wedge red) & \text{[implication]} \\
\equiv\ & big \vee (\neg soft \vee \neg red) & \text{[De Morgan]} \\
\equiv\ & (big \vee \neg soft \vee \neg red) & [\vee \text{ associativity}]
\end{aligned}
$$

7. An Example Resolution Proof

CNF Knowledge Base KB:

(1)	$(big \vee \neg soft \vee \neg red)$
(2)	$(soft \vee \neg new)$
(3)	(new)
(4)	(red)

Proposition to prove:

$$big$$

Resolution proof:

(5)	$(\neg big)$	[assumption]
(6)	$(\neg soft \vee \neg red)$	[by (1),(5)]
(7)	$(\neg new \vee \neg red)$	[by (2),(6)]
(8)	$(\neg red)$	[by (3),(7)]
(9)	$\perp$	[by (4),(8)]

So by resolution soundness:

$$KB \models big$$

[index]

8. Comparing Resolution and Prolog Proofs

Resolution proof (from Slide 7):

(5) $(\neg big)$

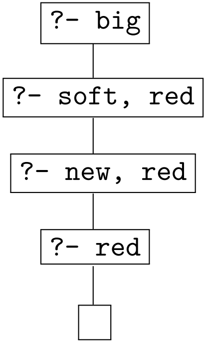
(6) $(\neg soft \lor \neg red)$

(7) $(\neg new \lor \neg red)$

(8) $(\neg red)$

(9) $\perp$

Prolog search tree (from Slide 5):



[index]

9. The Meaning of Prolog Programs with Negation-as-Failure

Negation-as-failure ("\+") cannot be directly translated as classical logic negation ("¬"). A simple example shows why:

Prolog program:

```
:- dynamic new/0, soft/0, red/0.
```

```
soft :- \+ new, red.
red.
```

Query:

```
?- soft.
true.
```

Equivalent knowledge base KB?

$$soft \leftarrow (\neg new \land red).$$
$$red.$$

No equivalent entailment:

$$KB \not\models soft$$

[index]

10. A Prolog Search Tree with Negation-as-Failure

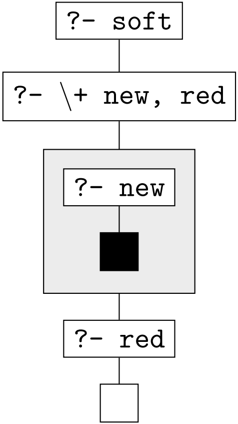
Prolog program:

```
:- dynamic new/0, soft/0, red/0.
```

```
soft :- \+ new, red.
red.
```

Query:

```
?- soft.
true.
```



[index]

11. Models of $\{soft \leftarrow (\neg new \wedge red),\ red\}$

Reminders:

- In propositional logic, the models of a formula are the lines in the formula's truth table that make the formula true.
- If F and G are formulas, then "$F \vDash G$" means "every model of F is also a model of G".

	$soft$	new	red	$(soft$	$\leftarrow$	$(\neg$	new	$\wedge$	$red))$	$\wedge$	red	
	t	t	t	$\cancel{t}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	t	$\cancel{t}$	
	t	t	f	$\cancel{t}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{f}$	f	$\cancel{f}$	
	t	f	t	$\cancel{t}$	$\cancel{t}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{t}$	t	$\cancel{t}$	
	t	f	f	$\cancel{t}$	$\cancel{t}$	$\cancel{t}$	$\cancel{f}$	$\cancel{f}$	$\cancel{f}$	f	$\cancel{f}$	
$\Longrightarrow$	f	t	t	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	t	$\cancel{t}$	$\Longleftarrow$
	f	t	f	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{f}$	f	$\cancel{f}$	
	f	f	t	$\cancel{f}$	$\cancel{f}$	$\cancel{t}$	$\cancel{f}$	$\cancel{t}$	$\cancel{t}$	f	$\cancel{t}$	
	f	f	f	$\cancel{f}$	$\cancel{t}$	$\cancel{t}$	$\cancel{f}$	$\cancel{f}$	$\cancel{f}$	f	$\cancel{f}$	

So, $\{soft \leftarrow (\neg new \wedge red),\ red\} \nvDash soft$ because line 5 is not a model of $soft$.

12. Clark Completion for Propositional Programs

Given a proposition ‘p’ appearing in a propositional program PR:

1. If ‘p’ does not appear as a fact or as the head of a clause in PR then $\mathbf{COMP}(\mathrm{p}, PR) = \neg p$.
2. If ‘p’ appears as a fact in PR then $\mathbf{COMP}(\mathrm{p}, PR) = p$.
3. If ‘p’ does not appear as a fact in PR and has a definition

   ```
   p :- Body_1.   ...   p :- Body_n.
   ```

   then $\mathbf{COMP}(\mathrm{p}, PR) = [p \leftrightarrow (B_1 \vee \ldots \vee B_n)]$, where each B_i is the same as ‘Body_i’ but with ‘\+’ replaced by ‘¬’, and ‘,’ replaced by ‘∧’.

Clark Completion:
$\mathbf{COMP}(PR) = \{\mathbf{COMP}(\mathrm{p}, PR) \mid \mathrm{p} \text{ appears in } PR\}$.

13. Clark Completion: Propositional Examples

PR	$\mathbf{COMP}(PR)$
`soft :- \+ new, red.` `red.`	$soft \leftrightarrow (\neg new \wedge red)$, red, $\neg new$

`big :- soft, red.`
`big :- \+ new.`
`red.`
	$big \leftrightarrow ((soft \wedge red) \vee \neg new)$,
	red,
	$\neg new$,
	$\neg soft$

`big :- soft.`	$big \leftrightarrow soft$, $\neg soft$

14. Clark Completion: Another Example

What is $\mathbf{COMP}(PR)$ if PR is the following program?

```
happy :- on_holiday, has_money.
happy :- work_done, \+ has_lectures.
has_money :- \+ student.
has_lectures :- term_time, student.
work_done.
```

14. Clark Completion: Another Example

What is $\mathbf{COMP}(PR)$ if PR is the following program?

```
happy :- on_holiday, has_money.
happy :- work_done, \+ has_lectures.
has_money :- \+ student.
has_lectures :- term_time, student.
work_done.
```

Answer: $\mathbf{COMP}(PR)$ is:

$$\begin{aligned}
&happy \leftrightarrow ((on_holiday \land has_money) \lor (work_done \land \neg has_lectures)),\\
&has_money \leftrightarrow \neg student,\\
&has_lectures \leftrightarrow (term_time \land student),\\
&work_done,\\
&\neg on_holiday,\\
&\neg student,\\
&\neg term_time
\end{aligned}$$

15. Prolog Soundness and Completeness

Let '`p`' be a proposition appearing in the propositional Prolog program PR.

Soundness:

If PR returns '`true`' for the query '`?- p`' then $\mathbf{COMP}(PR) \models p$.

If PR returns '`false`' for the query '`?- p`' then $\mathbf{COMP}(PR) \models \neg p$.

Completeness:

If $\mathbf{COMP}(PR) \models p$ and PR is *stratified* (i.e. does not contain loops) then PR will return '`true`' for the query '`?- p`'.

If $\mathbf{COMP}(PR) \models \neg p$ and PR is *stratified* (i.e. does not contain loops) then PR will return '`false`' for the query '`?- p`'.

16. Clark Completion for Predicate Prolog Programs

Clark completion can be applied to predicate Prolog programs as well, but extra steps and conditions must be applied:

- Programs must be *safe* – written in such a way that negative sub-goals (i.e. sub-goals with '\+') are evaluated only after the variables in them have all been *ground* (i.e. substituted with terms containing no variables).

- Clauses must be re-written in *general form*, i.e. including quantifiers and with only universally quantified variables in their head, before the $\mathbf{COMP}$ procedure is applied.

- *Clark equality theory* must be added to the completion to ensure that the '=' predicate corresponds to Prolog unification.

[index]

17. Safe and Unsafe Prolog Programs

An unsafe Prolog program:

```
happy(X) :-
        \+ sad(X).
sad(mani).
hungry(nina).
```

Queries:

```
?- happy(X).
false.
?- happy(nina).
true.
```

A safe Prolog program:

```
happy(X) :-
        person(X),
        \+ sad(X).
sad(mani).
hungry(nina).
person(mani).
person(nina).
```

Query:

```
?- happy(X).
X=nina.
```

[index]

18. Example General Form and Completion

Prolog program PR:

```
likes(X, partner_of(X)).
likes(nina, X) :- polite(X).
polite(partner_of(mani)).
```

PR in general form:

$$\forall x_1 \forall x_2.[\mathit{likes}(x_1, x_2) \leftarrow \exists x.(x_1 = x \wedge x_2 = \mathit{partner\_of}(x))],$$
$$\forall x_1 \forall x_2.[\mathit{likes}(x_1, x_2) \leftarrow \exists x.(x_1 = \mathit{nina} \wedge x_2 = x \wedge \mathit{polite}(x))],$$
$$\forall x_1.[\mathit{polite}(x_1) \leftarrow x_1 = \mathit{partner\_of}(\mathit{mani})]$$

$\mathbf{COMP}(PR)$:

$$\forall x_1 \forall x_2.[\mathit{likes}(x_1, x_2) \leftrightarrow [\exists x.(x_1 = x \wedge x_2 = \mathit{partner\_of}(x)) \vee \exists x.(x_1 = \mathit{nina} \wedge x_2 = x \wedge \mathit{polite}(x))]],$$
$$\forall x_1.[\mathit{polite}(x_1) \leftrightarrow x_1 = \mathit{partner\_of}(\mathit{mani})],$$

– *plus the Clark equality theory for* PR.

19. Example Completion with Equality Theory

Prolog program PR:

```
likes(X, partner_of(X)).
likes(nina, X) :- polite(X).
polite(partner_of(mani)).
```

$\mathbf{COMP}(PR)$:

$$\forall x_1 \forall x_2.[\mathit{likes}(x_1, x_2) \leftrightarrow [\exists x.(x_1 = x \wedge x_2 = \mathit{partner\_of}(x)) \vee \exists x.(x_1 = \mathit{nina} \wedge x_2 = x \wedge \mathit{polite}(x))]],$$

$$\forall x_1.[\mathit{polite}(x_1) \leftrightarrow x_1 = \mathit{partner\_of}(\mathit{mani})],$$

<u>Clark equality theory for PR:</u>

$\mathit{mani} \neq \mathit{nina} \wedge \forall x.[\mathit{partner\_of}(x) \neq \mathit{mani} \wedge \mathit{partner\_of}(x) \neq \mathit{nina}]$,

$\forall x_1 \forall x_2.[x_1 \neq x_2 \rightarrow \mathit{partner\_of}(x_1) \neq \mathit{partner\_of}(x_2)]$,

plus for any structured term $\tau[x]$ containing any variable x:

$\tau[x] \neq x$ [e.g. $\mathit{partner\_of}(\mathit{partner\_of}(x)) \neq x$]

[index]

20. Summary

- A query execution with a stratified (i.e. non-looping) propositional Prolog program without negation-as-failure corresponds to a search for a resolution proof by contradiction.
- For "sensibly written" (i.e. safe, stratified) Prolog programs with negation-as-failure, their logical meaning can be understood as their Clark completion.
- For predicate Prolog programs, the Clark completion must include Clark equality theory, because of the way Prolog unifies terms and variables.
- In A.I. terms, Clark completion is related to the "closed world assumption", and Clark equality theories are examples of "uniquness-of-names axioms".

[index]

21. Further Reading for Technical Details

- Marek Sergot's notes on negation as failure are available at: https://www.doc.ic.ac.uk/~mjs/teaching/KnowledgeRep491/NBF_491-2x1.pdf
  (Marek Sergot is a professor in the Computer Science Department at Imperial College London.)

- Keith Clark's original paper "Negation as Failure", first published in: Logic and Data Bases, (eds Gallaire and Minker), 1978, can be found at: https://www.doc.ic.ac.uk/~klc/NegAsFailure.pdf

[index]