Knowledge Representation and Reasoning

Logics for Artificial Intelligence

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu

Contents

2.3 Clause Form Propositional Logic

2.3.1 Clause Form Syntax part 1

Atomic Propositions:

- Any letter of the alphabet
- Any letter with a numeric subscript
- Any alphanumeric string.

Literals:

If P is an atomic proposition, P and $\neg P$ are literals.
P is called a **positive literal**
$\neg P$ is called a **negative literal**.

2.3.1 Clause Form Syntax part 2

Clauses: If $L_1, \ldots, L_n$ are literals
then the set $\{L_1, \ldots, L_n\}$ is a clause.

Sets of Clauses: If $C_1, \ldots, C_n$ are clauses
then the set $\{C_1, \ldots, C_n\}$ is a set of clauses.

2.3.2 Clause Form Semantics
Atomic Propositions

Intensional: $[P]$ is some proposition in the domain.

Extensional: $[\![P]\!]$ is either True or False.

2.3.2 Clause Form Semantics
Literals

Positive Literals: The meaning of P as a literal is the same as it is as an atomic proposition.

Negative Literals:

Intensional:
$[\neg P]$ means that it is not the case that $[P]$.

Extensional: $[\![\neg P]\!]$ is True if $[\![P]\!]$ is False;
Otherwise, it is False.

2.3.2 Clause Form Semantics

Clauses

Intensional:

$[\{L_1, \ldots, L_n\}] = [L_1]$ and/or ... and/or $[L_n]$.

Extensional:

$[\![\{L_1, \ldots, L_n\}]\!]$ is True
if at least one of $[\![L_1]\!]$, ..., $[\![L_n]\!]$ is True;
Otherwise, it is False.

2.3.2 Clause Form Semantics

Sets of Clauses

Intensional:

$[\{C_1, \ldots, C_n\}] = [C_1]$ and ... and $[C_n]$.

Extensional:

$[\![\{C_1, \ldots, C_n\}]\!]$ is True if $[\![C_1]\!]$ and ... and $[\![C_n]\!]$ are all True;

Otherwise, it is False.

Clause Form Proof Theory: Resolution

Notion of Proof: A set of clauses constitutes a proof.

Assumptions: The proof is initialized with a set of assumption clauses $AC_1, \ldots, AC_n$.

Rule of Inference: A clause may be added to a set of clauses if justified by **resolution**.

Derived Clause: If clause CQ has been added to a set of clauses initialized with the set of assumption clauses $AC_1, \ldots, AC_n$ by one or more applications of resolution, then $AC_1, \ldots, AC_n \vdash CQ$.

Resolution

$$\frac{\{P, L_1, \ldots, L_n\}, \{\neg P, L_{n+1}, \ldots, L_m\}}{\{L_1, \ldots, L_n, L_{n+1}, \ldots, L_m\}}$$

Resolution is sound, but not complete!

Example Derivation

1.	$\{\neg(TomIsTheDriver), \neg(TomIsThePassenger)\}$	*Assumption*
2.	$\{(TomIsThePassenger), (BettyIsThePassenger)\}$	*Assumption*
3.	$\{(TomIsTheDriver)\}$	*Assumption*
4.	$\{\neg(TomIsThePassenger)\}$	*R,1,3*
5.	$\{(BettyIsThePassenger)\}$	*R,2,4*

Example of Incompleteness

$\{P\} \models \{P, Q\}$

but

Resolution does not apply to $\{\{P\}\}$.

Resolution Refutation

- Notice that $\{\{P\}, \{\neg P\}\}$ is contradictory.

- Notice that resolution applies to $\{P\}$ and $\{\neg P\}$ producing $\{\}$, the **empty clause**.

- If a set of clauses is contradictory, repeated application of resolution is **guaranteed** to produce $\{\}$.

Implications

- $\{P_1, \ldots, P_n, Q_1, \ldots, Q_m\}$ is contradictory.

- means $(P_1 \wedge \ldots \wedge P_n \wedge Q_1 \wedge \ldots \wedge Q_m)$ is False in all models.

- means whenever $(P_1 \wedge \ldots \wedge P_n)$ is True, $(Q_1 \wedge \ldots \wedge Q_m)$ is False.

- means whenever $(P_1 \wedge \ldots \wedge P_n)$ is True $\neg(Q_1 \wedge \ldots \wedge Q_m)$ is True.

- means $P_1, \ldots, P_n \models \neg(Q_1 \wedge \ldots \wedge Q_m)$.

Negation and Clauses

- $\neg\{L_1, \ldots, L_n\} = \{\{\neg L_1\}, \ldots, \{\neg L_n\}\}$.

- $\neg L = \begin{cases} \neg A & \text{if } L = A \\ A & \text{if } L = \neg A \end{cases}$

Resolution Refutation

To decide if $C_1, \ldots, C_n \models CQ$:

1. Let $S = \{C_1, \ldots, C_n\} \cup \neg CQ$
2. Repeatedly apply resolution to clauses in S.
3. If generate $\{\}$, $C_1, \ldots, C_n \models CQ$.
4. If reach point where no new clause can be generated, but $\{\}$ has not appeared, $C_1, \ldots, C_n \not\models CQ$.

Example 1

To decide if $\{P\} \models \{P, Q\}$

$S = \{\{P\}, \{\neg P\}, \{\neg Q\}\}$

$$\begin{array}{lll} 1. & \{P\} & Assumption \\ 2. & \{\neg P\} & From\ query\ clause \\ 3. & \{\} & R, 1, 2 \end{array}$$

Example 2

To decide if

$$\{\neg(TomIsTheDriver), \neg(TomIsThePassenger)\},$$
$$\{(TomIsThePassenger), (BettyIsThePassenger)\},$$
$$\{(TomIsTheDriver)\} \qquad \models \{(BettyIsThePassenger)\}$$

1.	$\{\neg(TomIsTheDriver), \neg(TomIsThePassenger)\}$	$Assumption$
2.	$\{(TomIsThePassenger), (BettyIsThePassenger)\}$	$Assumption$
3.	$\{(TomIsTheDriver)\}$	$Assumption$
4.	$\{\neg(BettyIsThePassenger)\}$	$From\ query\ clause$
5.	$\{(TomIsThePassenger)\}$	$R, 2, 4$
6.	$\{\neg(TomIsTheDriver)\}$	$R, 1, 5$
7.	$\{\}$	$R, 3, 6$

Resolution Efficiency Rules

Tautology Elimination: If clause C contains literals L and $\neg L$, delete C from the set of clauses.

Pure-Literal Elimination: If clause C contains a literal A ($\neg A$) and no clause contains a literal $\neg A$ (A), delete C from the set of clauses.

Subsumption Elimination: If the set of clauses contains clauses C_1 and C_2 such that $C_1 \subseteq C_2$, delete C_2 from the set of clauses.

These rules delete unhelpful clauses.

Resolution Strategies

Unit Preference: Resolve shorter clauses before longer clauses.

Set of Support: One clause in each pair being resolved must descend from the query.

Many others

These are heuristics for finding {} faster.

Example 1 Using `prover`

```
prover(6): (prove '(P) '(P or Q))

 1  (P)              Assumption
 2  ((~ P))          From Query
 3  ((~ Q))          From Query
 4  nil              R  2  1 nil
QED
```

Example 2 Using `prover`

```
prover(8): (prove '(((~ TomIsTheDriver) or (~ TomIsThePassenger))
                    (TomIsThePassenger or BettyIsThePassenger)
                    TomIsTheDriver)
                  'BettyIsThePassenger)
 1  (TomIsTheDriver)  Assumption
 2  ((~ TomIsTheDriver) (~ TomIsThePassenger))  Assumption
 3  (TomIsThePassenger BettyIsThePassenger) Assumption
 4  ((~ BettyIsThePassenger)) From Query
 5  (TomIsThePassenger) R  4  3 nil
Deleting  3 (TomIsThePassenger BettyIsThePassenger)
because it's subsumed by  5 (TomIsThePassenger)
 6  ((~ TomIsTheDriver))  R  5  2 nil
Deleting  2 ((~ TomIsTheDriver) (~ TomIsThePassenger))
because it's subsumed by  6 ((~ TomIsTheDriver))
 7  nil             R  6  1 nil
QED
```

Example 1 Using SNARK

```
snark-user(29): (assert 'P)
nil

snark-user(30): (prove '(or P Q))

(Refutation
(Row 1
   P
   assertion)
(Row 2
   false
   (rewrite ~conclusion 1))
)
:proof-found
```

Properties of Resolution Refutation

Resolution Refutation is sound, complete, and a decision procedure

for Clause Form Propositional Logic.

It remains so when Tautology Elimination, Pure-Literal Elimination, Subsumption and the Unit-Preference Strategy are included.

It remains so when Set of Support is used as long as the assumptions are not contradictory.

Translating Standard Wfps into Clause Form

Useful Meta-Theorem: The Subformula Property

If A is (an occurrence of) a subformula of B,

and $\models A \Leftrightarrow C$,

then $\models B \Leftrightarrow B\{C/A\}$

Translating Standard Wfps into Clause Form Step 1

Eliminate occurrences of $\Leftrightarrow$ using

$$\models (A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$$

From: $(LivingThing \Leftrightarrow (Animal \vee Vegetable))$

To:
$((LivingThing \Rightarrow (Animal \vee Vegetable))$
$\wedge((Animal \vee Vegetable) \Rightarrow LivingThing))$

Translation Step 2

Eliminate occurrences of $\Rightarrow$ using

$$\models (A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$$

From:
$((LivingThing \Rightarrow (Animal \vee Vegetable))$
$\wedge((Animal \vee Vegetable) \Rightarrow LivingThing))$

To:
$((\neg LivingThing \vee (Animal \vee Vegetable))$
$\wedge(\neg(Animal \vee Vegetable) \vee LivingThing))$

Translation Step 3

Translate to *miniscope* form using

$$\models \neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$$
$$\models \neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$$
$$\models \neg(\neg A) \Leftrightarrow A$$

From:

$((\neg LivingThing \vee (Animal \vee Vegetable))$
$\wedge(\neg(Animal \vee Vegetable) \vee LivingThing))$

To:

$((\neg LivingThing \vee (Animal \vee Vegetable))$
$\wedge((\neg Animal \wedge \neg Vegetable) \vee LivingThing))$

Translation Step 4

CNF: Translate into Conjunctive Normal Form, using

$$\models (A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$$
$$\models (A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$$

From:

$((\neg LivingThing \vee (Animal \vee Vegetable))$

$\wedge((\neg Animal \wedge \neg Vegetable) \vee LivingThing))$

To:

$((\neg LivingThing \vee (Animal \vee Vegetable))$

$\wedge((\neg Animal \vee LivingThing) \wedge (\neg Vegetable \vee LivingThing)))$

Translation Step 5

Discard extra parentheses using the associativity and commutativity of $\wedge$ and $\vee$.

From:

$$((\neg LivingThing \vee (Animal \vee Vegetable))$$
$$\wedge((\neg Animal \vee LivingThing) \wedge (\neg Vegetable \vee LivingThing)))$$

To:

$$((\neg LivingThing \vee Animal \vee Vegetable)$$
$$\wedge(\neg Animal \vee LivingThing)$$
$$\wedge(\neg Vegetable \vee LivingThing))$$

Translation Step 6

Turn each disjunction into a clause,
and the conjunction into a set of clauses.

From:

$((\neg LivingThing \vee Animal \vee Vegetable)$
$\wedge(\neg Animal \vee LivingThing)$
$\wedge(\neg Vegetable \vee LivingThing))$

To:

$((\neg LivingThing\ \ Animal\ \ Vegetable)$
$(\neg Animal\ \ LivingThing)$
$(\neg Vegetable\ \ LivingThing))$

Use of Translation

$$A_1, \ldots, A_n \models_{Standard} B$$

iff

The translation of $A_1 \wedge \cdots \wedge A_n \wedge \neg B$ into a set of clauses is contradictory.

Connections

Modus Ponens

$$\frac{A, A \Rightarrow B}{B}$$

Resolution

$$\frac{\{A\}, \{\neg A, B\}}{\{B\}}$$

Modus Tollens

$$\frac{A \Rightarrow B, \neg B}{\neg A}$$

Resolution

$$\frac{\{\neg A, B\}, \{\neg B\}}{\{\neg A\}}$$

Disjunctive Syllogism

$$\frac{A \vee B, \neg A}{B}$$

Resolution

$$\frac{\{A, B\}, \{\neg A\}}{\{B\}}$$

Chaining

$$\frac{A \Rightarrow B, B \Rightarrow C}{A \Rightarrow C}$$

Resolution

$$\frac{\{\neg A, B\}, \{\neg B, C\}}{\{\neg A, C\}}$$

More Connections

Clause	**Rule**
$\{\neg A_1, \ldots, \neg A_n, C_1, \ldots, C_m\}$	$(A_1 \wedge \cdots \wedge A_n) \Rightarrow (C_1 \vee \cdots \vee C_m)$
Horn Clause	**Rule**
$\{\neg A_1, \ldots, \neg A_n, C\}$	$(A_1 \wedge \cdots \wedge A_n) \Rightarrow C$
	Prolog Clause
	C :- $A_1, \ldots, A_n$
Set of Support	**Back-chaining**

prover Example

```
prover(57): (prove '((LivingThing <=> (Animal or Vegetable))
                     (LivingThing & (~ Animal)))
                   'Vegetable)

 1  (LivingThing)   Assumption
 2  ((~ Animal))    Assumption
 3  ((~ Animal) LivingThing)  Assumption
 4  ((~ Vegetable) LivingThing) Assumption
 5  ((~ LivingThing) Animal Vegetable)  Assumption
 6  ((~ Vegetable)) From Query

Deleting  3 ((~ Animal) LivingThing)
because it's subsumed by  1 (LivingThing)
Deleting  4 ((~ Vegetable) LivingThing)
because it's subsumed by  1 (LivingThing)
```

`prover` Example, continued

```
 1  (LivingThing)   Assumption
 2  ((~ Animal))    Assumption
 5  ((~ LivingThing) Animal Vegetable)  Assumption
 6  ((~ Vegetable)) From Query

 7  ((~ LivingThing) Animal)  R  6  5 nil
Deleting  5 ((~ LivingThing) Animal Vegetable)
because it's subsumed by  7 ((~ LivingThing) Animal)
 8  (Animal)         R  7  1 nil
 9  ((~ LivingThing)) R  7  2 nil
10  nil              R  9  1 nil
QED
```