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

Part III

3 Predicate Logic Over Finite Models

3.1 CarPool World

Propositional Logic

Tom drives Betty	*Betty drives Tom*
Tom is the driver	*Betty is the driver*
Tom is the passenger	*Betty is the passenger*

related only by the domain rules.

Predicate Logic

$Drives(Tom, Betty)$	$Drives(Betty, Tom)$
$Driver(Tom)$	$Driver(Betty)$
$Passenger(Tom)$	$Passenger(Betty)$

shows two properties, one relation, and two individuals.

3.2 The “Standard” Finite-Model Predicate Logic

3.2.1 Syntax of the “Standard” Finite-Model Predicate Logic Atomic Symbols

Individual Constants:

- Any letter of the alphabet (preferably early),
- any (such) letter with a numeric subscript,
- any character string not containing blanks nor other punctuation marks.

For example: a, B_{12}, Tom, *Tom’s_mother-in-law*.

Atomic Symbols, Part 2

Variables:

- Any letter of the alphabet (preferably late),
- any (such) letter with a numeric subscript.

For example: u, v_6.

Atomic Symbols, Part 3

Predicate Symbols:

- Any letter of the alphabet (preferably late middle),
- any (such) letter with a numeric subscript,
- any character string not containing blanks.

For example: $P, Q_4, Drives$.

Each Predicate Symbol must have a particular **arity**.

Use superscript for explicit arity.

For example: $P^1, Drives^2, Q_2^3$

Atomic Symbols, Part 4

In any specific predicate logic language

Individual Constants,
Variables,
Predicate Symbols

must be disjoint.

Set of individual constants and of predicate symbols must be **finite**.

Terms

- Every individual constant and variable is a term.
- Nothing else is a term.

Atomic Formulas

If P^n is a predicate symbol of arity n,

and $t_1, \ldots, t_n$ are terms,

then $P^n(t_1, \ldots, t_n)$ is an atomic formula.

E.g.: $\mathit{Passenger}^{\mathit{1}}(\mathit{Tom}), \mathit{Drives}^{\mathit{2}}(\mathit{Betty}, y)$

(The superscript may be omitted if no confusion results.)

Well-Formed Formulas (wffs):

Every atomic formula is a wff.

If P is a wff, then so is $(\neg P)$.

If P and Q are wffs, then so are

$$(P \wedge Q) \qquad (P \vee Q)$$

$$(P \Rightarrow Q) \quad (P \Leftrightarrow Q)$$

If P is a wff and x is a variable,
then $\forall x(P)$ and $\exists x(P)$ are wffs.

Parentheses may be omitted or replaced by square brackets if no confusion results.

We will allow $(P_1 \wedge \cdots \wedge P_n)$ and $(P_1 \vee \cdots \vee P_n)$.

$\forall x(\forall y(P))$ may be abbreviated as $\forall x, y(P)$.
$\exists x(\exists y(P))$ may be abbreviated as $\exists x, y(P)$.

Quantifiers:

In $\forall x P$ and $\exists x P$

$\forall$ called the **universal quantifier**.

$\exists$ called the **existential quantifier**.

P is called the **scope** of quantification.

Free and Bound Variables

Every occurrence of x in P, not in the scope of some other occurrence of $\forall x$ or $\exists x$, is said to be **free** in P and **bound** in $\forall xP$ and $\exists xP$.

Every occurrence of every variable other than x that is free in P is also free in $\forall xP$ and $\exists xP$.

$$\forall x[P(x,y) \Leftrightarrow [(\exists x \exists z Q(x,y,z)) \Rightarrow R(x,y)]]$$

Open, Closed, and Ground

A wff with a free variable is called **open**.

A wff with no free variables is called **closed**,

An expression with no variables is called **ground**.

CarPool World Domain Rules

PropositionalLogic

$Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger$

$Tom\ is\ the\ driver \Leftrightarrow \neg Tom\ is\ the\ passenger$

$Betty\ drives\ Tom \Rightarrow Betty\ is\ the\ driver \wedge Tom\ is\ the\ passenger$

$Tom\ drives\ Betty \Rightarrow Tom\ is\ the\ driver \wedge Betty\ is\ the\ passenger$

$Tom\ drives\ Betty \vee Betty\ drives\ Tom$

PredicateLogic

$\forall x(Driver(x) \Leftrightarrow \neg Passenger(x))$

$\forall x, y(Drives(x, y) \Rightarrow (Driver(x) \wedge Passenger(y)))$

$Drives(Tom, Betty) \vee Drives(Betty, Tom)$

3.2.2 Substitutions
Syntax

Pairs: t/v (Read : “t for v”)

- t is any term
- v is any variable

Substitutions: $\{t_1/v_1, \ldots, t_n/v_n\}$

- $i \neq j \Rightarrow v_i \neq v_j$

Terminology

$\sigma = \{t_1/v_1, \ldots, t_n/v_n\}$

t_i is a **term** in σ

v_i is a **variable of** σ

Say $t_i/v_i \in \sigma$ and $v_i \in \sigma$,
but not $t_i \in \sigma$

Note: x is not a variable of $\{x/y\}$,
i.e. $x/y \in \{x/y\}, y \in \{x/y\}, x \notin \{x/y\}$

Substitution Application

For expression $\mathcal{A}$ and substitution $\sigma = \{t_1/v_1, \ldots, t_n/v_n\}$

$\mathcal{A}\sigma$: replace every free occurrence of each v_i in $\mathcal{A}$ by t_i

E.g.:

$P(x, y)\{x/y, y/x\} = P(y, x)$

$\forall x[P(x, y) \Leftrightarrow [(\exists x \exists z Q(x, y, z)) \Rightarrow R(x, y, z)]]\{a/x, b/y, c/z\}$
$= \forall x[P(x, b) \Leftrightarrow [(\exists x \exists z Q(x, b, z)) \Rightarrow R(x, b, c)]]$

3.2.3 Semantics of Finite-Model Predicate Logic

Assumes a **Finite Domain**, $\mathcal{D}$, of

- individuals,
- sets of individuals,
- relations over individuals

Let $\mathcal{I}$ be the set of all individuals in $\mathcal{D}$.

Semantics of Individual Constants

$[a] = [\![a]\!] =$ some particular individual in $\mathcal{I}$.

There is no anonymous individual.
I.e. for every individual, i in $\mathcal{I}$, there is an individual constant c such that $[c] = [\![c]\!] =$ i.

Semantics of Predicate Symbols

Predicate Symbols:

- $[P^1]$ is some category/property of individuals of $\mathcal{I}$.
- $[P^n]$ is some n-ary relation over $\mathcal{I}$.
- $[\![P^1]\!]$ is some particular subset of $\mathcal{I}$.
- $[\![P^n]\!]$ is some particular subset of the relation

$$\underbrace{\mathcal{I} \times \cdots \times \mathcal{I}}_{n \text{ times}}.$$

Intensional Semantics of Ground Atomic Formulas

- If P^1 is some unary predicate symbol, and t is some individual constant, then $[P^1(t)]$ is the proposition that $[t]$ is an instance of the category $[P^1]$ (or has the property $[P^1]$).
- If P^n is some n-ary predicate symbol, and $t_1, \ldots, t_n$ are individual constants, then $[P^n(t_1, \ldots, t_n)]$ is the proposition that the relation $[P^n]$ holds among individuals $[t_1]$, and $\ldots$, and $[t_n]$.

Extensional Semantics of Ground Atomic Formulas

- If P^1 is some unary predicate symbol,
  and t is some individual constant,
  then $[\![P^{\mathit{1}}(t)]\!]$ is True if $[\![t]\!] \in [\![P^{\mathit{1}}]\!]$,
  and False otherwise.

- If P^n is some n-ary predicate symbol,
  and $t_1, \ldots, t_n$ are individual constants,
  then $[\![P^n(t_{\mathit{1}}, \ldots, t_n)]\!]$ is True
  if $\langle [\![t_{\mathit{1}}]\!], \ldots, [\![t_n]\!] \rangle \in [\![P^n]\!]$,
  and False otherwise.

Semantics of WFFs, Part 1

$[\neg P]$, $[P \wedge Q]$, $[P \vee Q]$, $[P \Rightarrow Q]$, $[P \Leftrightarrow Q]$
$[\![\neg P]\!]$, $[\![P \wedge Q]\!]$, $[\![P \vee Q]\!]$, $[\![P \Rightarrow Q]\!]$, and $[\![P \Leftrightarrow Q]\!]$
are as they are in Propositional Logic.

Semantics of WFFs, Part 2

- $[\forall x P]$ is the proposition that every individual i in $\mathcal{I}$, with "name" t_i, satisfies $[P\{t_i/x\}]$.
- $[\exists x P]$ is the proposition that some individual i in $\mathcal{I}$, with "name" t_i, satisfies $[P\{t_i/x\}]$.
- $[\![\forall x P]\!]$ is True if $[\![P\{t/x\}]\!]$ is True for every individual constant, t. Otherwise, it is False.
- $[\![\exists x P]\!]$ is True if there is some individual constant, t such that $[\![P\{t/x\}]\!]$ is True. Otherwise, it is False.

Intensional Semantics of Individual Constants In CarPool World

$[\mathit{Tom}]$ = Someone named Tom.
$[\mathit{Betty}]$ = Someone named Betty.

Intensional Semantics of Individual Constants In 4-Person CarPool World (Call it 4pCarPool World)

$[Tom]$ = Someone named Tom.

$[Betty]$ = Someone named Betty.

$[John]$ = Someone named John.

$[Mary]$ = Someone named Mary.

Intensional Semantics of Ground Atomic Wffs In Both CarPool Worlds

Predicate Symbols:

$[\mathit{Driver}^{1}(x)] = [x]$ is the driver of the/a car.
$[\mathit{Passenger}^{1}(x)] = [x]$ is the passenger of the/a car.
$[\mathit{Drives}^{2}(x, y)] = [x]$ drives $[y]$ to work.

Extensional Semantics of One CarPool World Situation

$[\![Tom]\!] = \text{Tom}.$

$[\![Betty]\!] = \text{Betty}.$

$[\![Driver]\!] = \{\text{Betty}\}.$

$[\![Passenger]\!] = \{\text{Tom}\}.$

$[\![Drives]\!] = \{\langle \text{ Betty, Tom}\rangle\}.$

Extensional Semantics of One 4pCarPool World Situation

$[\![Tom]\!] = \text{Tom}.$

$[\![Betty]\!] = \text{Betty}.$

$[\![John]\!] = \text{John}.$

$[\![Mary]\!] = \text{Mary}.$

$[\![Driver]\!] = \{\text{Betty, John}\}.$

$[\![Passenger]\!] = \{\text{Mary, Tom}\}.$

$[\![Drives]\!] = \{\langle \text{ Betty, Tom}\rangle, \langle \text{ John, Mary}\rangle\}.$

3.2.4 Model Checking in Finite-Model Predicate Logic

- n Individual Constants.
- Predicate P^j yields n^j ground atomic propositions.
- k_j predicates of arity j yields $\sum_j (k_j \times n^j)$ ground atomic propositions.
- So $2^{\sum_j (k_j \times n^j)}$ situations (columns of truth table).
- CarPool World has $2^{(2\times 2^1+1\times 2^2)} = 2^8 = 256$ situations.
- 4pCarPool World has $2^{(2\times 4^1+1\times 4^2)} = 2^{24} = 16,777,216$ situations.

Some CarPool World Situations

$Driver(Tom)$	T	T	F
$Driver(Betty)$	T	F	T
$Passenger(Tom)$	T	F	T
$Passenger(Betty)$	T	T	F
$Drives(Tom, Tom)$	T	F	F
$Drives(Tom, Betty)$	T	T	F
$Drives(Betty, Tom)$	T	F	T
$Drives(Betty, Betty)$	T	F	F
$\forall x(Driver(x) \Leftrightarrow \neg Passenger(x))$	F	T	T
$\forall x \forall y(Drives(x, y) \Rightarrow (Driver(x) \Leftrightarrow Passenger(y)))$	T	T	T

Turning Predicate Logic Over Finite Domains Into Ground Predicate Logic

If $c_1, \ldots, c_n$ are the individual constants,

- Turn $\forall x P(x)$ into $P(c_1) \wedge \cdots \wedge P(c_n)$
- and $\exists x P(x)$ into $P(c_1) \vee \cdots \vee P(c_n)$
- E.g.:

$$
\begin{aligned}
&\forall x \exists y (Drives(x, y)) \\
&\Leftrightarrow \exists y Drives(Tom, y) \wedge \exists y Drives(Betty, y) \\
&\Leftrightarrow (Drives(Tom, Tom) \vee Drives(Tom, Betty)) \\
&\quad \wedge (Drives(Betty, Tom) \vee Drives(Betty, Betty))
\end{aligned}
$$

Sorted Logic: A Digression

Introduce a hierarchy of **sorts**, $s_1, \ldots, s_n$.

(A sort in logic is similar to a data type in programming.)

Assign each individual constant a **sort**.

Assign each variable a **sort**.

Declare the sort of each argument position of each predicate symbol.

An atomic formula, $P^n(t_1, \ldots, t_n)$ is only syntactically valid if the sort of t_i, for each i, is the sort, or a subsort of the sort, declared for the i^{th} argument position of P^n.

Predicate 2-Car CarPool World in Decreasoner

```
sort commuter
commuter Tom, Betty
sort car
car TomsCar, BettysCar

;;; [DrivesIn(x,y,c)] = [x] drives [y] to work in car [c].
predicate DrivesIn(commuter,commuter,car)

;;; [DriverOf(x,c)] = [x] is the driver of car [c].
predicate DriverOf(commuter,car)

;;; [PassengerIn(x,c)] = [x] is a passenger in car [c].
predicate PassengerIn(commuter,car)
```

Number of Ground Atomic Propositions Unsorted vs. Sorted

Atomic Proposition	Unsorted	Sorted
DrivesIn(commuter,commuter,car)	$4^3 = 64$	$2^3 = 8$
DriverOf(commuter,car)	$4^2 = 16$	$2^2 = 4$
PassengerIn(commuter,car)	$4^2 = 16$	$2^2 = 4$
Total	96	16

Domain Rules of 2-Car CarPool World

```
/projects/shapiro/CSE563/decreasoner/examples/ShapiroCSE563/4cCPWPRedRules.e

;;; If someone's a driver of one car, they're not a passenger in any car.
;;; (And if someone's a passenger in one car, they're not driver of any car.)
[commuter][car1][car2](DriverOf(commuter,car1) -> !PassengerIn(commuter,car2)).

;;; If A drives B in car C, then A is the driver of and B is a passenger in C.
[commuter1][commuter2][car](DrivesIn(commuter1,commuter2,car)
                            -> DriverOf(commuter1,car)
                               & PassengerIn(commuter2,car)).

;;; Either Tom drives Betty in Tom's car or Betty drives Tom in Betty's car.
DrivesIn(Tom,Betty,TomsCar) | DrivesIn(Betty,Tom,BettysCar).

;;; Tom doesn't drive Betty's car, and Betty doesn't drive Tom's car.
!DriverOf(Tom,BettysCar) & !DriverOf(Betty,TomsCar).

;;; Neither Tom nor Betty is a passenger in their own car.
!PassengerIn(Tom,TomsCar) & !PassengerIn(Betty,BettysCar).
```

Decreasoner Produces Two Models

The True propositions:

```
model 1:                            model 2:
DriverOf(Betty, BettysCar).         DriverOf(Tom, TomsCar).
DrivesIn(Betty, Tom, BettysCar).    DrivesIn(Tom, Betty, TomsCar)
PassengerIn(Tom, BettysCar).        PassengerIn(Betty, TomsCar).
```

Use of Predicate-Wang

```
cl-user(12): (wang:predicate-entails
               '( (forall (x y)
                          (if (Drives x y)
                              (and (Driver x) (Passenger y))))
                 (Drives Betty Tom))
               '(and (Driver Betty) (Passenger Tom))
               '(Betty Tom))
t
```

3.3 Clause Form
Finite-Model Predicate Logic

3.3.1 Syntax of Clause Form Finite-Model Predicate Logic

Individual constants, predicate symbols, terms, and ground atomic formulas as in standard finite-model predicate logic.
(Variables are not needed.)

Literals, clauses and sets of clauses as in propositional clause form logic.

3.3.2 Semantics of Clause Form Finite-Model Predicate Logic

- Individual constants, predicate symbols, terms, and ground atomic formulas as in standard finite-model predicate logic.
- Ground literals, ground clauses, and sets of ground clauses as in propositional clause form logic.

Translation of Standard Form to Clause Form Finite-Model Predicate Calculus

1. Eliminate quantifiers as when using model checking.
2. Translate into clause form as for propositional logic.

3.3.3 Model Finding: GSAT

```
procedure GSAT($C$, $tries$, $flips$)
    input: a set of clauses $C$, and positive integers $tries$ and $flips$
    output: a model satisfying $C$, or failure
for $i := 1$ to $tries$ do
    $\mathcal{M}$ := a randomly generated truth assignment
    for $j := 1$ to $flips$ do
        if $\mathcal{M} \models C$ then return $\mathcal{M}$
        $p$ := an atom such that a change in its truth
               assignment gives the largest increase in the total
               number of clauses in $C$ that are satisfied by $\mathcal{M}$
        $\mathcal{M} := \mathcal{M}$ with the truth assignment of $p$ reversed
    end for end for
return "no satisfying interpretation found"
```

[Brachman & Levesque, p. 82–83, based on Bart Selman, Hector J. Levesque and David Mitchell, A New Method for Solving Hard Satisfiability Problems, AAAI-92.]

A Pedagogical Implementation of GSAT

`/projects/shapiro/CSE563/gsat.cl`

Uses `wang:expand` to eliminate quantifiers,

and `prover:clauseForm` to translate to clause form.

Example GSAT Run

```
cl-user(1): :ld /projects/shapiro/CSE563/gsat
...
cl-user(2): :pa gsat
gsat(3): (gsat '((forall x (iff (Driver x) (not (Passenger x))))
                  (forall (x y) (if (Drives x y) (and (Driver x) (Passenger y))))
                  (or (Drives Tom Betty) (Drives Betty Tom))
                  (Driver Betty))
                30 6)

 A satisfying model (found on try 17) is
(((Driver Tom) nil)            ((Passenger Tom) t)
 ((Drives Betty Betty) nil)    ((Drives Tom Tom) nil)
 ((Drives Betty Tom) t)        ((Drives Tom Betty) nil)
 ((Driver Betty) t)            ((Passenger Betty) nil))
#<equal hash-table with 8 entries @ #x4a64dca>
```

Using GSAT to Find The Value of a Wff in a KB

```
gsat(19): (ask '(and (Drives Betty Tom) (Passenger Tom))
               '((forall x (iff (Driver x) (not (Passenger x))))
                 (forall (x y) (if (Drives x y) (and (Driver x) (Passenger y)
                 (or (Drives Tom Betty) (Drives Betty Tom))
                 (Driver Betty))
               30 6)

 A satisfying model (found on try 19) is
(((Drives Tom Tom) nil)     ((Drives Betty Tom) t)
 ((Driver Betty) t)         ((Passenger Tom) t)
 ((Drives Tom Betty) nil)   ((Driver Tom) nil)
 ((Drives Betty Betty) nil) ((Passenger Betty) nil))

(and (Drives Betty Tom) (Passenger Tom)) is True in a model of the KB.
nil
```

Model Finding: Walksat
A More Efficient Version of GSAT

DIMACS FORMAT:

Code each atomic formula as a positive integer:

```
c 1 Drives(Tom, Betty) Tom drives Betty to work.
c 2 Drives(Betty, Tom) Betty drives Tom to work.
c 3 Driver(Tom) Tom is the driver of the car.
c 4 Driver(Betty) Betty is the driver of the car.
c 5 Passenger(Tom) Tom is the passenger of the car.
c 6 Passenger(Betty) Betty is the passenger of the car.
```

DIMACS cont'd

Code each clause as a set ± integers, terminated by 0:

```
c ((~ (Driver Tom)) (~ (Passenger Tom)))
-3 -5 0
c ((~ (Driver Betty)) (~ (Passenger Betty)))
-4 -6 0
c ((Passenger Tom) (Driver Tom))
5 3 0
c ((Passenger Betty) (Driver Betty))
6 4 0
c ((~ (Drives Tom Betty)) (Driver Tom))
-1 3 0
c ((~ (Drives Betty Tom)) (Driver Betty))
-2 4 0
c ((~ (Drives Tom Betty)) (Passenger Betty))
-1 6 0
c ((~ (Drives Betty Tom)) (Passenger Tom))
-2 5 0
c ((Drives Tom Betty) (Drives Betty Tom))
1 2 0
c ((Driver Betty))
4 0
```

Running Walksat

```
% /projects/shapiro/CSE563/WalkSAT/Walksat_v46/walksat -solcnf
    < /projects/shapiro/CSE563/WalkSAT/cpw.cnf
...
ASSIGNMENT FOUND
v -1
v 2
v -3
v 4
v 5
v -6
```

Model Finding: Decreasoner

Decreasoner translates sorted finite-model predicate logic wffs into DIMACS clause form.

Decreasoner gives set of clauses to Relsat.

Relsat systematically searches all models. It either:

> reports that there are no satisfying models;
> returns up to `MAXMODELS` (currently 100) satisfying models;
> or gives up.

If Relsat gives up, Decreasoner gives set of clauses to Walksat. It either:

> returns some satisfying models;
> or returns some “near misses”;
> or gives up.

Decreasoner, Walksat, and “Near Misses”

“Let’s say that an ”N-near miss model of a SAT problem” is a truth assignment that satisfies all but N clauses of the problem. Walksat provides the command-line option:

`-target N` = succeed if N or fewer clauses unsatisfied

If relsat produces no models, the Discrete Event Calculus Reasoner invokes walksat with `-target 1`. If this fails, it invokes walksat with `-target 2`. If this fails, it gives up. One or two unsatisfied clauses may be helpful for debugging. In my experience, three or more unsatisfied clauses are less useful.

If you get a near miss model, it’s often useful to rerun the Discrete Event Calculus Reasoner. Because walksat is stochastic, you may get back a different near miss model, and that near miss model may be more informative than the previous one.”

[Erik Mueller, email to scs, 1/12/2007]