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 I

Part II

Part III

1 Introduction

1.1 Knowledge Representation

Artificial Intelligence (AI)

A field of computer science and engineering concerned with the computational understanding of what is commonly called intelligent behavior, and with the creation of artifacts that exhibit such behavior.

Knowledge Representation

A subarea of Artificial Intelligence concerned with understanding, designing, and implementing ways of representing information in computers so that programs (agents) can use this information

- to derive information that is implied by it,
- to converse with people in natural languages,
- to decide what to do next
- to plan future activities,
- to solve problems in areas that normally require human expertise.

Reasoning

Deriving information that is implied by the information already present is a form of reasoning.

Knowledge representation schemes are useless without the ability to reason with them.

So, Knowledge Representation and Reasoning (KRR)

Manifesto of KRR

a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows…In order for a program to be capable of learning something it must first be capable of being told it. John McCarthy, "Programs with Common Sense", 1959.

Knowledge *vs.* Belief

Knowledge: justified true belief.

John believes that the world is flat: Not true.

Sally believes that the first player in chess can always win,
Betty believes that the second player can always win,
and Mary believes that, with optimal play on both sides, chess will always end in a tie.
One of them is correct,
but none are justified.

So *Belief* Representation & Reasoning: more accurate
But we'll continue to say KRR.

In Class Exercise

“An Approach to Serenity”

Easy NL Inferences

Every student studies hard.
Therefore every smart student studies.

Tuesday evening, Jack either went to the movies, played bridge, or studied.
Tuesday evening, Jack played bridge.
Therefore, Jack neither went to the movies nor studied Tuesday evening.

Background Knowledge: Some Sentences and How We Understand Them.

John likes ice cream.

John likes to eat ice cream.

Mary likes Asimov.

Mary likes to read books written by Isaac Asimov.

Background Knowledge: Some Sentences and How We Understand Them.

Bill flicked the switch.
The room was flooded with light.
Bill moved the switch to the “on” position, which caused a light to come on, which lit up the room Bill was in.

Betty opened the blinds.
The courtyard was flooded with light.
Betty adjusted the blinds so that she could see through the window they were in front of, after which she could see that the courtyard on the other side of the window was bright.

Memory Integration in Humans

After seeing these sentences (among others),

The sweet jelly was on the kitchen table.
The ants in the kitchen ate the jelly.
The ants ate the sweet jelly that was on the table.
The sweet jelly was on the table.
The jelly was on the table.
The ants ate the jelly.

subjects, with high confidence reported that they had seen the sentence,
The ants ate the sweet jelly that was on the kitchen table.

[Bransford and Franks (1971). The abstraction of linguistic ideas. *Cognitive Psychology, 2,* 331-350, as reported on `http://www.rpi.edu/~verwyc/cognotes5.htm`.]

Requirements for a Knowledge-Based Agent

1. *"what it already knows"* [McCarthy '59]
   A knowledge base of beliefs.
2. *"it must first be capable of being told"* [McCarthy '59]
   A way to put new beliefs into the knowledge base.
3. *"automatically deduces for itself a sufficiently wide class of immediate consequences"* [McCarthy '59]
   A reasoning mechanism to derive new beliefs from ones already in the knowledge base.

1.2 Logic

- **Logic** is the study of correct reasoning.
- It is not a particular KRR language.
- There are many systems of logic (logics).
- AI KRR research can be seen as a hunt for the “right” logic.

Commonalities among Logics

- System for reasoning.
- Prevent reasoning from “truths” to “falsities”.
  (But can reason from truths and falsities to truths and falsities.)
- Language for expressing reasoning steps.

Parts of the Study/Specification of a Logic

Syntax: The atomic symbols of the logical language, and the rules for constructing well-formed, nonatomic expressions (symbol structures) of the logic.

Semantics: The meanings of the atomic symbols of the logic, and the rules for determining the meanings of nonatomic expressions of the logic.

Proof Theory: The rules for determining a subset of logical expressions, called **theorems** of the logic.

2 Propositional Logic

Logics that do not analyze information below the level of the proposition.

2.1 What is a Proposition?

An expression in some language

- that is true or false
- whose negation makes sense
- that can be believed or not
- whose negation can be believed or not
- that can be put in the frame
  "I believe that it is not the case that __________."

Examples

Of propositions

- Betty is the driver of the car.
- Barack Obama is sitting down or standing up.
- If Opus is a penguin, then Opus doesn't fly.

Of non-propositions

- Barack Obama
- how to ride a bicycle
- If the fire alarm rings, leave the building.

Sentences *vs.* Propositions

A sentence is an expression of a (written) language that begins with a capital letter and ends with a period, question mark, or exclamation point.

Some sentences do not contain a proposition:
“Hi!”, “Why?”, “Pass the salt!”

Some sentences do not express a proposition, but contain one:
“Is Betty driving the car?”

Some sentences contain more than one proposition:
If Opus is a penguin, then Opus doesn’t fly.

2.2 CarPool World: A Motivational "Micro-World"

- Tom and Betty carpool to work.
- On any day, either Tom drives Betty or Betty drives Tom.
- In the former case, Tom is the driver and Betty is the passenger.
- In the latter case, Betty is the driver and Tom is the passenger.

Propositions:	Betty drives Tom.	Tom drives Betty.
	Betty is the driver.	Tom is the driver.
	Betty is the passenger.	Tom is the passenger.

2.3 The “Standard” Propositional Logic

2.3.1 Syntax of the “Standard” Propositional Logic

Atomic Propositions

- Any letter of the alphabet, e.g.: P
- Any letter of the alphabet with a numerical subscript, e.g.: Q_3
- Any alphanumeric string, e.g.: *Tom is the driver*

is an atomic proposition.

Well-Formed Propositions (WFPs)

1. Every atomic proposition is a wfp.
2. If P is a wfp, then so is $(\neg P)$.
3. If P and Q are wfps, then so are

   (a) $(P \wedge Q)$ (b) $(P \vee Q)$

   (c) $(P \Rightarrow Q)$ (d) $(P \Leftrightarrow Q)$
4. Nothing else is a wfp.

Parentheses may be omitted. Precedence: $\neg$; $\wedge$, $\vee$; $\Rightarrow$; $\Leftrightarrow$.
Will allow $(P_1 \wedge \cdots \wedge P_n)$ and $(P_1 \vee \cdots \vee P_n)$.
Square brackets may be used instead of parentheses.

Examples of WFPs

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

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

$$Betty\ drives\ Tom \Leftrightarrow \neg Tom\ is\ the\ driver$$

Alternative Symbols

$\neg$: $\sim$ `!`

$\wedge$: $\&$ $\cdot$

$\vee$: $|$

$\Rightarrow$: $\rightarrow$ $\supset$ `->`

$\Leftrightarrow$: $\leftrightarrow$ $\equiv$ `<->`

A Computer-Readable Syntax for Wfps

Based on CLIF, the Common Logic Interchange Format[a]

Atomic Propositions: Use one of:

- **Embedded underscores:** `Betty_drives_Tom`
- **Embedded hyphens:** `Betty-drives-Tom`
- **CamelCase:** `BettyDrivesTom`
- **sulkingCamelCase:** `bettyDrivesTom`
- **Escape brackets:** `|Betty drives Tom|`
- **Quotation marks:** `"Betty drives Tom"`

---

[a] ISO/IEC, Information technology — Common Logic (CL): a framework for a family of logic-based languages, ISO/IEC 24707:2007(E), 2007.

CLIF for Non-Atomic Wfps

Print Form	**CLIF Form**
$\neg P$	`(not P)`
$P \wedge Q$	`(and P Q)`
$P \vee Q$	`(or P Q)`
$P \Rightarrow Q$	`(if P Q)`
$P \Leftrightarrow Q$	`(iff P Q)`
$(P_1 \wedge \cdots \wedge P_n)$	`(and P1 ...Pn)`
$(P_1 \vee \cdots \vee P_n)$	`(or P1 ...Pn)`

Semantics of Atomic Propositions 1
Intensional Semantics

- Dependent on a Domain.
- Independent of any specific interpretation/model/possible world/situation.
- Statement in a previously understood language (e.g. English) that allows truth value to be determined in any specific situation.
- Often omitted, but shouldn't be.

Intensional CarPool World Semantics

$[\textit{Betty drives Tom}]$ = Betty drives Tom to work.

$[\textit{Tom drives Betty}]$ = Tom drives Betty to work.

$[\textit{Betty is the driver}]$ = Betty is the driver of the car.

$[\textit{Tom is the driver}]$ = Tom is the driver of the car.

$[\textit{Betty is the passenger}]$ = Betty is the passenger in the car.

$[\textit{Tom is the passenger}]$ = Tom is the passenger in the car.

Alternative Intensional CarPool World Semantics

$[\textit{Betty drives Tom}]$ = Tom drives Betty to work.

$[\textit{Tom drives Betty}]$ = Betty drives Tom to work.

$[\textit{Betty is the driver}]$ = Tom is the passenger in the car.

$[\textit{Tom is the driver}]$ = Betty is the passenger in the car.

$[\textit{Betty is the passenger}]$ = Tom is the driver of the car.

$[\textit{Tom is the passenger}]$ = Betty is the driver of the car.

Alternative CarPool World
Syntax/Intensional Semantics

$[A]$ = Betty drives Tom to work.

$[B]$ = Tom drives Betty to work.

$[C]$ = Betty is the driver of the car.

$[D]$ = Tom is the driver of the car.

$[E]$ = Betty is the passenger in the car.

$[F]$ = Tom is the passenger in the car.

Intensional Semantics Moral

- Don’t omit.
- Don’t presume.
- No “pretend it’s English semantics”.

Intensional Semantics of WFPs

$[\neg P]$ = It is not the case that $[P]$.

$[P \wedge Q] = [P]$ and $[Q]$.

$[P \vee Q]$ = Either $[P]$ or $[Q]$ or both.

$[P \Rightarrow Q]$ = If $[P]$ then $[Q]$.

$[P \Leftrightarrow Q] = [P]$ if and only if $[Q]$.

Example CarPool World Intensional WFP Semantics

$[\mathit{Betty\ drives\ Tom} \Leftrightarrow \neg \mathit{Tom\ is\ the\ driver}]$

$=$ Betty drives Tom to work

if and only if Tom is not the driver of the car.

Terminology

- $\neg P$ is called the **negation** of P.
- $P \wedge Q$ is called the **conjunction** of P and Q.
  P and Q are referred to as **conjuncts**.
- $P \vee Q$ is called the **disjunction** of P and Q.
  P and Q are referred to as **disjuncts**.
- $P \Rightarrow Q$ is called a **conditional** or **implication**.
  P is referred to as the **antecedent**;
  Q as the **consequent**.
- $P \Leftrightarrow Q$ is called a **biconditional** or **equivalence**.

2.3.2 Semantics of Atomic Propositions 2

Extensional Semantics

- Relative to an interpretation/model/possible world/situation.
- Either True or False.

Extensional CarPool World Semantics

	Denotation in Situation				
Proposition	1	2	3	4	5
Betty drives Tom	True	True	True	False	False
Tom drives Betty	True	True	False	True	False
Betty is the driver	True	True	True	False	False
Tom is the driver	True	False	False	True	False
Betty is the passenger	True	False	False	True	False
Tom is the passenger	True	False	True	False	False

Note: n propositions $\Rightarrow 2^n$ possible situations.

6 propositions in CarPool World
$\Rightarrow 2^6 = 64$ different situations.

Extensional Semantics of WFPs

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

$[\![P \wedge Q]\!]$ is True if $[\![P]\!]$ is True and $[\![Q]\!]$ is True. Otherwise, it is False.

$[\![P \vee Q]\!]$ is False if $[\![P]\!]$ is False and $[\![Q]\!]$ is False. Otherwise, it is True.

$[\![P \Rightarrow Q]\!]$ is False if $[\![P]\!]$ is True and $[\![Q]\!]$ is False. Otherwise, it is True.

$[\![P \Leftrightarrow Q]\!]$ is True if $[\![P]\!]$ and $[\![Q]\!]$ are both True, or both False. Otherwise, it is False.

Note that this is the outline of a recursive function that evaluates a wfp, given the truth values of its atomic propositions.

Extensional Semantics Truth Tables

P	True	False
$\neg P$	False	True

P	True	True	False	False
Q	True	False	True	False
$P \wedge Q$	True	False	False	False
$P \vee Q$	True	True	True	False
$P \Rightarrow Q$	True	False	True	True
$P \Leftrightarrow Q$	True	False	False	True

Notice that each column of these tables represents a different situation.

Material Implication

$P \Rightarrow Q$ is True when P is False.

So,
If the world is flat, then the moon is made of green cheese
is considered True if *if . . . then* is interpreted as material implication.

$$(P \Rightarrow Q) \Leftrightarrow (\neg P \vee Q)$$

P	True	True	False	False
Q	True	False	True	False
$\neg P$	False	False	True	True
$P \Rightarrow Q$	True	False	True	True
$\neg P \vee Q$	True	False	True	True

$(P \Rightarrow Q)$ is sometimes taken as a abbreviation of $(\neg P \vee Q)$

Note: “Uninterpreted Language”, **Formal** Logic, applicable to every logic in the class.

Example CarPool World Truth Table

Betty drives Tom	True	True	False	False
Tom is the driver	True	False	True	False
$\neg$*Tom is the driver*	False	True	False	True
Betty drives Tom $\Leftrightarrow \neg$*Tom is the driver*	False	True	True	False

Computing Denotations

Use the procedure sketched on page 41.

Use Spreadsheet:
See `http://www.cse.buffalo.edu/~shapiro/Courses/CSE563/truthTable.xls/`

Use `Boole` program from Barwise & Etchemendy package

Computing the Denotation of a Wfp in a Model

Construct a truth table containing all atomic wfps and the wfp whose denotation is to be computed, and restrict the truth table to the desired model.
E.g., play with `http://www.cse.buffalo.edu/~shapiro/Courses/CSE563/cpw.xls/`

Use the program `/projects/shapiro/CSE563/denotation`

Example Runs of `denotation` Program

```
cl-user(1): (denotation '(if p (if q p))
                        '((p . True) (q . False)))
True

cl-user(2): (denotation
             '(if BettyDrivesTom
                  (not TomIsThePassenger))
             '((BettyDrivesTom . True)
               (TomIsThePassenger . True)))
False
```

Model Finding

A model **satisfies** a wfp if the wfp is True in that model.

If a wfp P is False in a model, $\mathcal{M}$, then $\mathcal{M}$ satisfies $\neg P$.

A model satisfies a set of wfps if they are all True in the model.

A model, $\mathcal{M}$, satisfies the wfps $P_1, \ldots, P_n$ if and only if $\mathcal{M}$, satisfies $P_1 \wedge \ldots \wedge P_n$.

Task: Given a set of wfps, A, find **satisfying models**.
I.e., models that assign all wfps in A the value True.

Model Finding with a Spreadsheet

Play with `http://www.cse.buffalo.edu/~shapiro/Courses/CSE563/cpw.xls/`

An Informal Model Finding Algorithm (Exponential)

- Given: Wfps labeled True, False, or unlabeled.
- If any wfp is labeled both True and False, terminate with failure.
- If all atomic wfps are labeled, return labeling as a model.
- If $\neg P$ is
  - labeled True, try labeling P False.
  - labeled False, try labeling P True.
- If $P \wedge Q$ is
  - labeled True, try labeling P and Q True.
  - labeled False, try labeling P False, and try labeling Q False.

Model Finding Algorithm, cont'd

- If $P \vee Q$ is
  - labeled False, try labeling P and Q False.
  - labeled True, try labeling P True, and try labeling Q True.
- If $P \Rightarrow Q$ is
  - labeled False, try labeling P True and Q False.
  - labeled True, try labeling P False, and try labeling Q True.
- If $P \Leftrightarrow Q$ is
  - labeled True, try labeling P and Q both True, and try labeling P and Q both False.
  - labeled False, try labeling P True and Q False, and try labeling P False and Q True.

Tableau Procedure for Model Finding[a]

$$T : BP \Rightarrow \neg BD$$

$$T : TD \Rightarrow BP$$

$$F : \neg BD$$

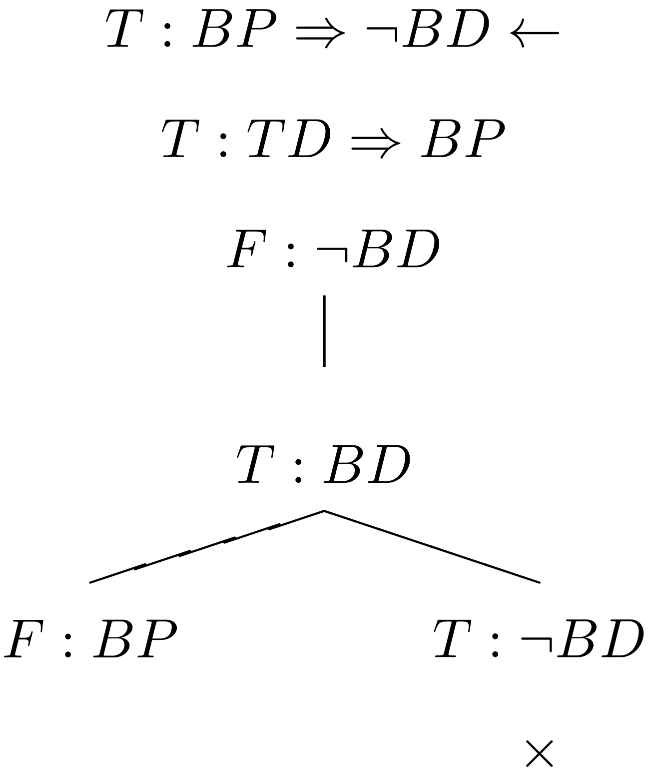
---

[a] Based on the semantic tableaux of Evert W. Beth, *The Foundations of Mathematics,* (Amsterdam: North Holland), 1959.

Tableau Procedure Example: Step 1

$$\begin{array}{c} T : BP \Rightarrow \neg BD \\[1em] T : TD \Rightarrow BP \\[1em] F : \neg BD \leftarrow \\ \Big| \\ T : BD \end{array}$$

Tableau Procedure Example: Step 2



$T : BP \Rightarrow \neg BD \leftarrow$

$T : TD \Rightarrow BP$

$F : \neg BD$

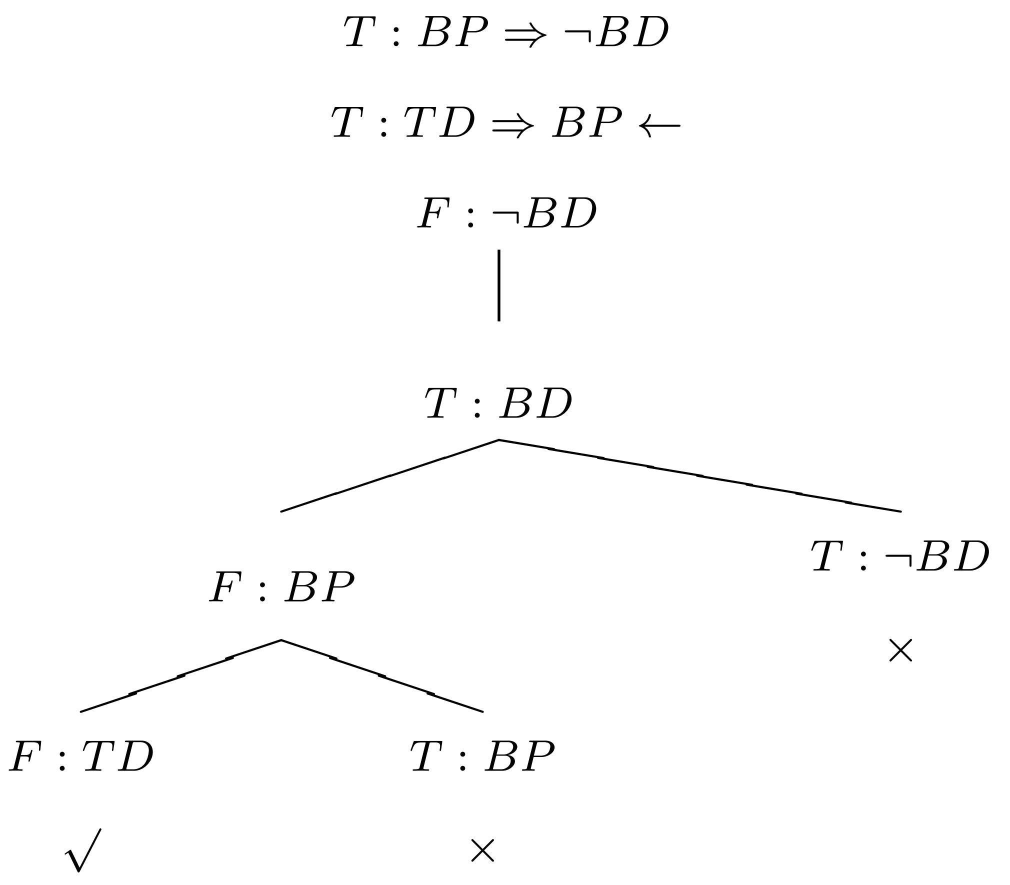
$T : BD$

$F : BP$

$T : \neg BD$

$\times$

Tableau Procedure Example: Step 3



$T : BP \Rightarrow \neg BD$

$T : TD \Rightarrow BP \leftarrow$

$F : \neg BD$

$T : BD$

$F : BP$

$T : \neg BD$

$\times$

$F : TD$

$T : BP$

$\surd$

$\times$

Model: $[\![BD]\!] = \text{True}$; $[\![BP]\!] = \text{False}$; $[\![TD]\!] = \text{False}$

Lisp Program for Tableau Procedure

Function: `(models trueWfps &optional falseWfps trueAtoms falseAtoms)`

```
<timberlake:~:1:62> mlisp
...
cl-user(1): :ld /projects/shapiro/CSE563/modelfinder
; Loading /projects/shapiro/CSE563/modelfinder.cl

cl-user(2): (models '( (if BP (not BD)) (if TD BP)) '((not BD)))
(((BD . True) (BP . False) (TD . False)))

cl-user(3): (models '( BDT (if BDT (and BD TP)) (not (or TP BD))))
nil

cl-user(4): (models '( (if BDT (and BD TP)) (if TDB (and TD BP))))
(((TD . True) (BP . True) (BD . True) (TP . True))
 ((BD . True) (TP . True) (TDB . False))
 ((TD . True) (BP . True) (BDT . False))
 ((BDT . False) (TDB . False)))
```

Decreasoner,[a] An Efficient Model Finder

On `nickelback.cse.buffalo.edu`
or `timberlake.cse.buffalo.edu`,
do

```
cd /projects/shapiro/CSE563/decreasoner
```

and try

```
python ubdecreasonerP.py examples/ShapiroCSE563/cpwProp.e
```

and

```
python ubdecreasonerP.py examples/ShapiroCSE563/cpwPropFindModels.e
```

---

[a] Decreasoner is by Erik T. Mueller, and uses relsat, by Roberto J. Bayardo Jr. and Robert C. Schrag, and walksat, by Bart Selman and Henry Kautz.

Decreasoner Example Input File

`/projects/shapiro/CSE563/decreasoner/examples/ShapiroCSE563/cpwPropFindModels.e`:

```
;;; Example of Finding Models for Some Wfp
;;;    In a SubDomain of Propositional Car Pool World
;;; Stuart C. Shapiro
;;; January 23, 2009

proposition BettyIsDriver ; Betty is the driver of the car.
proposition TomIsDriver ; Tom is the driver of the car.
proposition BettyIsPassenger ; Betty is the passenger in the car.

;;; A set of well-formed propositions to find models of within CPW
(BettyIsPassenger -> !BettyIsDriver).
(TomIsDriver -> BettyIsPassenger).
!!BettyIsDriver.
```

Decreasoner Example Run

```
<timberlake:decreasoner:1:60> python ubdecreasonerP.py
               examples/ShapiroCSE563/cpwPropFindModels.e

...
model 1:

BettyIsDriver.
!BettyIsPassenger.
!TomIsDriver.
```

Semantic Properties of WFPs

- A wfp is **satisfiable** if it is True in at least one situation.
- A wfp is **contingent** if it is True in at least one situation and False in at least one situation.
- A wfp is **valid** ($\models P$) if it is True in every situation.
  A valid wfp is also called a **tautology**.
- A wfp is **unsatisfiable** or **contradictory**
  if it is False in every situation.

Examples

P	True	True	False	False
Q	True	False	True	False
$\neg P$	False	False	True	True
$Q \Rightarrow P$	True	True	False	True
$P \Rightarrow (Q \Rightarrow P)$	True	True	True	True
$P \wedge \neg P$	False	False	False	False

$\neg P$, $Q \Rightarrow P$, and $P \Rightarrow (Q \Rightarrow P)$ are satisfiable,

$\neg P$ and $Q \Rightarrow P$ are contingent,

$P \Rightarrow (Q \Rightarrow P)$ is valid,

$P \wedge \neg P$ is contradictory.

Logical Entailment

$\{A_1, \ldots, A_n\}$ **logically entails** B in logic $\mathcal{L}$

$$A_1, \ldots, A_n \models_{\mathcal{L}} B$$

if B is True in every situation in which every A_i is True.

If $\mathcal{L}$ is assumed,

$$A_1, \ldots, A_n \models B$$

If $n = 0$, we have validity

$$\models B,$$

i.e., B is True in every situation.

Examples

P	True	True	False	False
Q	True	False	True	False
$\neg P$	False	False	True	True
$Q \Rightarrow P$	True	True	False	True
$P \Rightarrow (Q \Rightarrow P)$	True	True	True	True
$P \wedge \neg P$	False	False	False	False

$$\models P \Rightarrow (Q \Rightarrow P)$$

$$P \models Q \Rightarrow P$$

$$Q, Q \Rightarrow P \models P$$

A Metatheorem

$$A_1, \ldots, A_n \models B$$

iff

$$A_1 \wedge \cdots \wedge A_n \models B$$

Semantic Deduction Theorem (Metatheorem)

$A_1, \ldots, A_n \models P$ if and only if $\models A_1 \wedge \cdots \wedge A_n \Rightarrow P$.

So deciding validity and logical entailment are equivalent.

Domain Knowledge (Rules)

Used to reduce the set of situations to those that “make sense”.

Domain Rules for CarPool World

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

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

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

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

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

Sensible CarPool World Situations

The only 2 of the 64 in which all domain rules are True:

	Denotation in Situation	
Proposition	3	4
Betty drives Tom	True	False
Tom drives Betty	False	True
Betty is the driver	True	False
Tom is the driver	False	True
Betty is the passenger	False	True
Tom is the passenger	True	False
$\textit{Betty drives Tom} \Leftrightarrow \neg \textit{Tom is the driver}$	True	True

General Effect of Domain Rules

The number of models that satisfy a set of wfps is reduced (or stays the same) as the size of the set increases.

For a set of wfps, Γ, and a wfp P, if the number of models that satisfy $\Gamma \cup \{P\}$ is strictly less than the number of models that satisfy Γ, then P is **independent** of Γ.

Computer Tests of CPW Domain Rules

Spreadsheet: `http://www.cse.buffalo.edu/~shapiro/Courses/CSE563/cpwRules.xls`

Decreasoner (on nickelback or timberlake):

```
cd /projects/shapiro/CSE563/decreasoner
python ubdecreasonerP.py examples/ShapiroCSE563/cpwPropRules.e
```

CarPool World Domain Rules in Decreasoner

```
proposition BettyDrivesTom ; Betty drives Tom to work.
proposition TomDrivesBetty ; Tom drives Betty to work.
proposition BettyIsDriver ; Betty is the driver of the car.
proposition TomIsDriver ; Tom is the driver of the car.
proposition BettyIsPassenger ; Betty is the passenger in the car.
proposition TomIsPassenger ; Tom is the passenger in the car.

;;; CPW Domain Rules
BettyIsDriver <-> !BettyIsPassenger.
TomIsDriver <-> !TomIsPassenger.
BettyDrivesTom -> BettyIsDriver & TomIsPassenger.
TomDrivesBetty -> TomIsDriver & BettyIsPassenger.
TomDrivesBetty | BettyDrivesTom.
```

Decreasoner on CPW with Domain Rules

```
python ubdecreasonerP.py examples/ShapiroCSE563/cpwPropRules.e
...
model 1:

BettyDrivesTom.
BettyIsDriver.
TomIsPassenger.
!BettyIsPassenger.
!TomDrivesBetty.
!TomIsDriver.
---

model 2:

BettyIsPassenger.
TomDrivesBetty.
TomIsDriver.
!BettyDrivesTom.
!BettyIsDriver.
!TomIsPassenger.
```

The KRR Enterprise
(Propositional Logic Version)

Given a domain you are interested in reasoning about:

1. List the set of propositions (expressed in English) that captures the basic information of interest in the domain.

2. Formalize the domain by creating one atomic wfp for each proposition listed in step (1). List the atomic wfps, and, for each, show the English proposition as its intensional semantics.

The KRR Enterprise, Part 2

3. Using the atomic wfps, determine a set of domain rules so that all, but only, the situations of the domain that make sense satisfy them. Strive for a set of domain rules that is small and independent.

4. Optionally, formulate an additional set of situation-specific wfps that further restrict the domain to the set of situations you are interested in. We will call this restricted domain the “subdomain”.

5. Letting Γ be the set of domain rules plus situation-specific wfps, and A be any proposition you are interested in, A is True in the subdomain if $\Gamma \models A$, is false in the subdomain if $\Gamma \models \neg A$, and otherwise is True in some more specific situations of the subdomain, and False in others.

Computational Methods for Determining Entailment and Validity

Version 1

```
(defun entails (KB Q)
  "Returns t if the knowledge base KB entails the query Q;
    else returns nil."
  (loop for model in (models KB)
      unless (denotation Q model)
      do (return-from entails nil))
  t)
```

Two problems:

1. `models` does not really return all the satisfying models;
2. `entails` does extra work.

Tableau Methods
Model-Finding Refutation

To Show $A_1, \ldots, A_n \models P$:

- Try to find a model that satisfies $A_1, \ldots, A_n$ but falsifies P.
- If you succeed, $A_1, \ldots, A_n \not\models P$.
- If you fail, $A_1, \ldots, A_n \models P$.

All refutation model-finding methods are commonly called “tableau methods”.

Semantic Tableaux and Wang’s Algorithm are two tableau methods that are **decision procedures** for logical entailment in Propositional Logic.

Semantic Tableaux[a]
A Model-Finding Refutation Procedure

The semantic tableau refutation procedure is the same as the tableau model-finding procedure we saw earlier, except it uses model finding refutation to show $A_1, \ldots, A_n \models P$.

The goal is that all branches be closed.

---

[a] Evert W. Beth, *The Foundations of Mathematics,* (Amsterdam: North Holland), 1959.

A Semantic Tableau to Prove $TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$

$$T : TD$$

$$T : TD \Rightarrow BP$$

$$T : BP \Rightarrow \neg BD$$

$$F : \neg BD$$

A Semantic Tableau to Prove $TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$

$$T : TD$$

$$T : TD \Rightarrow BP$$

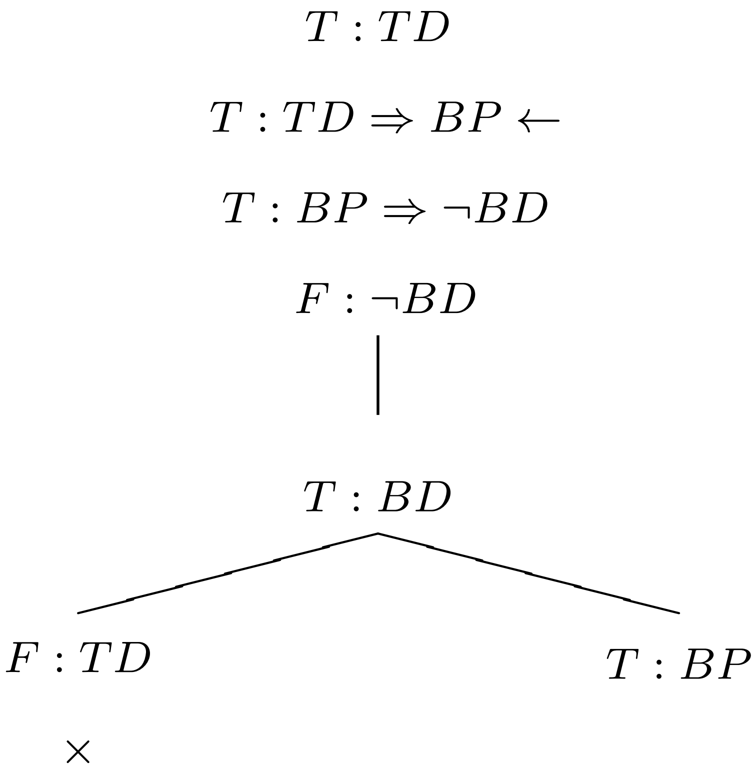
$$T : BP \Rightarrow \neg BD$$

$$F : \neg BD \leftarrow$$

$$|$$

$$T : BD$$

A Semantic Tableau to Prove $TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$



$T : TD$

$T : TD \Rightarrow BP \leftarrow$

$T : BP \Rightarrow \neg BD$

$F : \neg BD$

$T : BD$

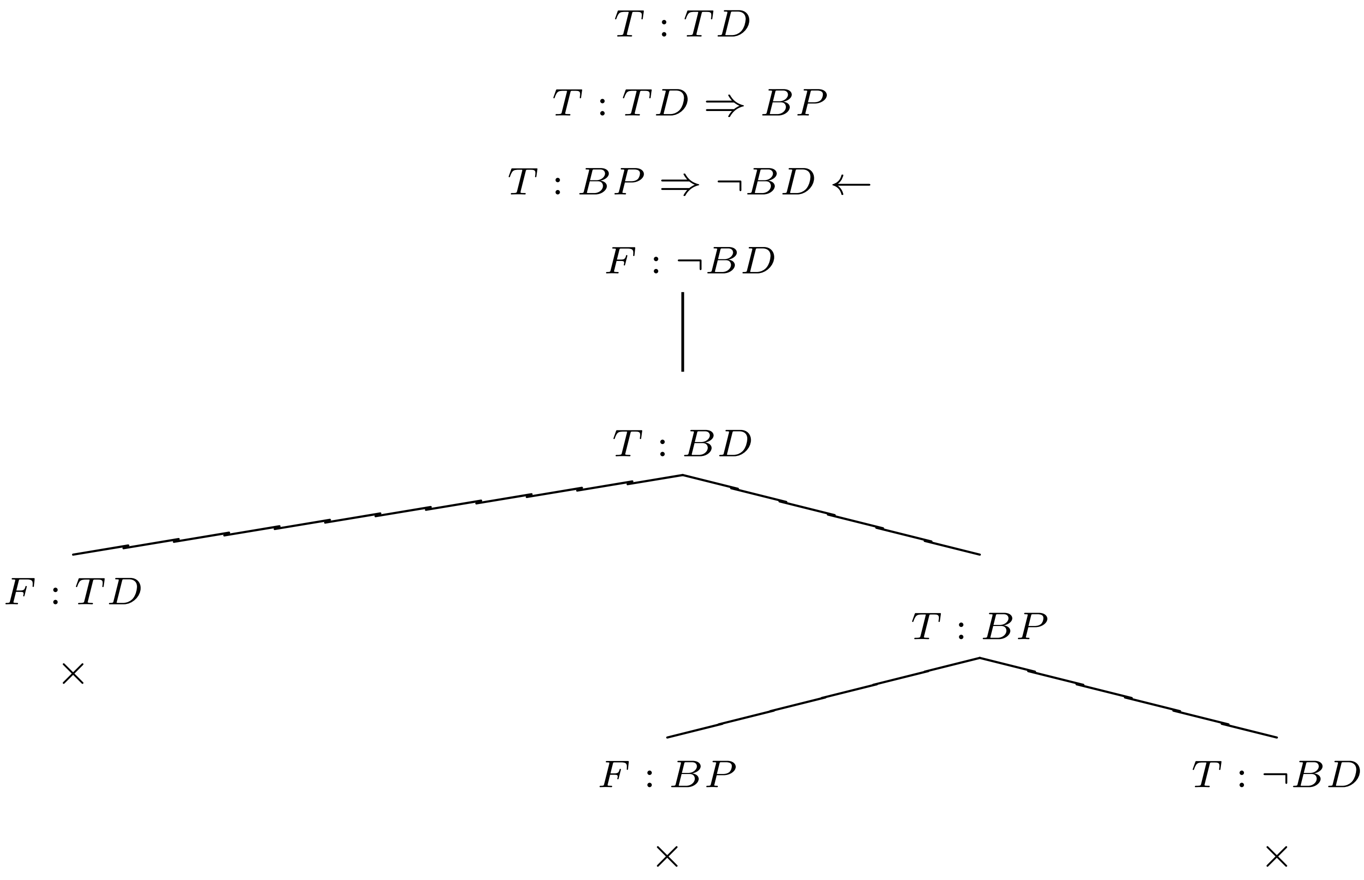
$F : TD$

$T : BP$

×

A Semantic Tableau To Prove

$$TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$$



$T : TD$

$T : TD \Rightarrow BP$

$T : BP \Rightarrow \neg BD \leftarrow$

$F : \neg BD$

$T : BD$

$F : TD$

×

$T : BP$

$F : BP$

×

$T : \neg BD$

×

A Semantic Tableau to Prove $TD \Rightarrow BP, BP \Rightarrow \neg BD \not\models \neg BD$

$$T : TD \Rightarrow BP$$

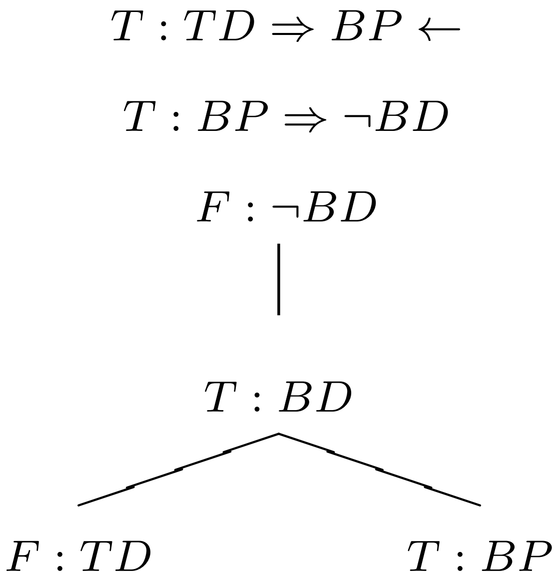
$$T : BP \Rightarrow \neg BD$$

$$F : \neg BD$$

A Semantic Tableau to Prove $TD \Rightarrow BP, BP \Rightarrow \neg BD \not\models \neg BD$

$$\begin{array}{c}
T : TD \Rightarrow BP \\[1em]
T : BP \Rightarrow \neg BD \\[1em]
F : \neg BD \leftarrow \\
\Big| \\[1em]
T : BD
\end{array}$$

A Semantic Tableau to Prove $TD \Rightarrow BP, BP \Rightarrow \neg BD \not\models \neg BD$



$T : TD \Rightarrow BP \leftarrow$

$T : BP \Rightarrow \neg BD$

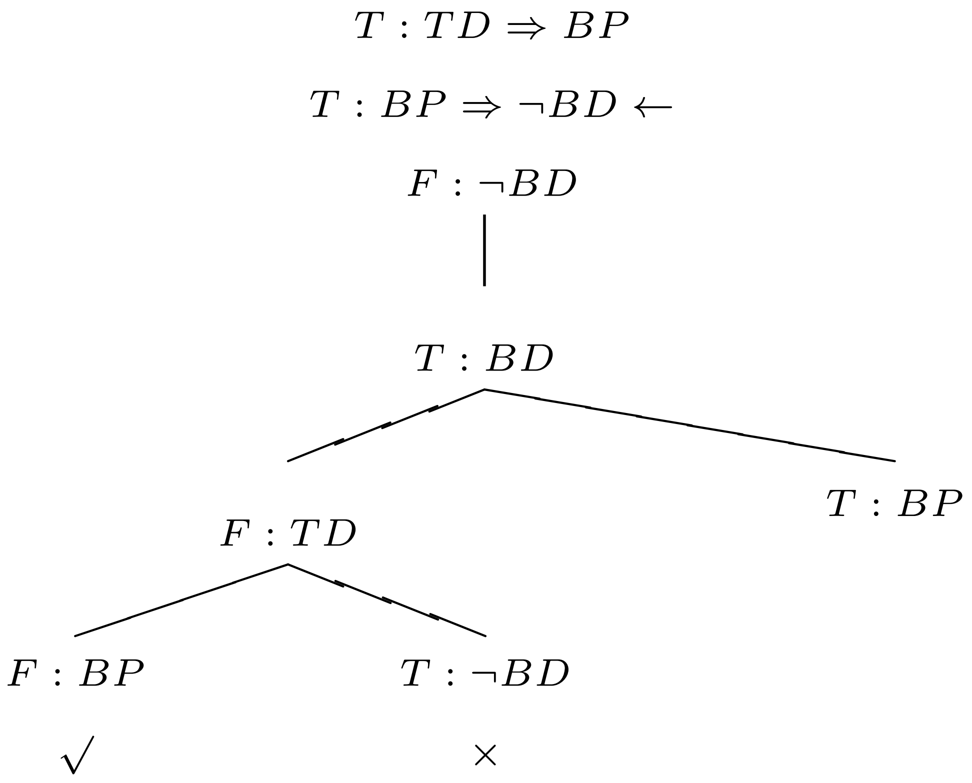
$F : \neg BD$

$T : BD$

$F : TD$

$T : BP$

A Semantic Tableau to Prove $TD \Rightarrow BP, BP \Rightarrow \neg BD \not\models \neg BD$



$T : TD \Rightarrow BP$

$T : BP \Rightarrow \neg BD \leftarrow$

$F : \neg BD$

$T : BD$

$F : TD$

$T : BP$

$F : BP$

$T : \neg BD$

$\surd$

$\times$

Can stop as soon as one satisfying model has been found.

Wang's Algorithm[a]

A Model-Finding Refutation Procedure

```
wang(Twfps, Fwfps) {
   /*
    * Twfps and Fwfps are sets of wfps.
    * Returns True if there is no model
    * that satisfies Twfps and falsifies Fwfps;
    * Otherwise, returns False.
   */
```

Note: is a version of `models`, but returns the opposite value.

---

[a] Hao Wang, Toward Mechanical Mathematics. IBM Journal of Research and Development 4, (1960), 2-22. Reprinted in K. M. Sayre and F. J. Crosson (Eds.) The Modeling of Mind: Computers and Intelligence. Simon and Schuster, New York, 1963.

Wang Algorithm

```
if $Twfps$ and $Fwfps$ intersect, return True;
if every $A \in Twfps \cup Fwfps$ is atomic, return False;

if $(P = (not\ \ A)) \in Twfps$,
  return $\mathrm{wang}(Twfps \setminus \{P\}, Fwfps \cup \{A\})$;
if $(P = (not\ \ A)) \in Fwfps$,
  return $\mathrm{wang}(Twfps \cup \{A\}, Fwfps \setminus \{P\})$;
```

Wang Algorithm

```
if $(P = (and\ \ A\ \ B)) \in Twfps$,
    return wang($(Twfps \setminus \{P\}) \cup \{A, B\}$, $Fwfps$);
if $(P = (and\ \ A\ \ B)) \in Fwfps$,
    return wang($Twfps$, $(Fwfps \setminus \{P\}) \cup \{A\}$)
               and wang($Twfps$, $(Fwfps \setminus \{P\}) \cup \{B\}$);

if $(P = (or\ \ A\ \ B)) \in Twfps$,
    return wang($(Twfps \setminus \{P\}) \cup \{A\}$, $Fwfps$);
               and wang($(Twfps \setminus \{P\}) \cup \{B\}$, $Fwfps$);
if $(P = (or\ \ A\ \ B)) \in Fwfps$,
    return wang($Twfps$, $(Fwfps \setminus \{P\}) \cup \{A, B\}$)
```

Wang Algorithm

```
if $(P = (\mathit{if}\ \ A\ \ B)) \in \mathit{Twfps}$,
    return wang$(\mathit{Twfps} \setminus \{P\},\ \mathit{Fwfps} \cup \{A\})$
            and wang($(\mathit{Twfps} \setminus \{P\}) \cup \{B\}$, $\mathit{Fwfps}$);
if $(P = (\mathit{if}\ \ A\ \ B)) \in \mathit{Fwfps}$,
    return wang$(\mathit{Twfps} \cup \{A\},\ (\mathit{Fwfps} \setminus \{P\}) \cup \{B\})$;

if $(P = (\mathit{iff}\ \ A\ \ B)) \in \mathit{Twfps}$,
    return wang$((\mathit{Twfps} \setminus \{P\}) \cup \{A, B\},\ \mathit{Fwfps})$
            and wang($\mathit{Twfps} \setminus \{P\}$, $\mathit{Fwfps} \cup \{A, B\}$);
if $(P = (\mathit{iff}\ \ A\ \ B)) \in \mathit{Fwfps}$,
    return wang$(\mathit{Twfps} \cup \{A\},\ (\mathit{Fwfps} \setminus \{P\}) \cup \{B\})$
            and wang($\mathit{Twfps} \cup \{B\}$, $(\mathit{Fwfps} \setminus \{P\}) \cup \{A\}$);
```

Implemented Wang Function

`(wang '(`$A_1, \ldots, A_n$`) '(`P`))`

Returns `t` if $A_1, \ldots, A_n \models P$;

`nil` otherwise.

Alternative View of Wang Function

`(wang` *KB* `(`*Query*`))`
Returns `t` if the *Query* follows from the *KB*
`nil` otherwise.

Front end:

`(entails` *KB* *Query*`)`
Returns `t` if the *Query* follows from the *KB*
`nil` otherwise.

Using Wang's Algorithm on a Tautology

```
(entails '() '(if A (if B A)))
 0[2]: (wang     nil   ((if A (if B A))))
   1[2]: (wang   (A)   ((if B A)))
     2[2]: (wang (B A) (A))
     2[2]: returned t
   1[2]: returned t
 0[2]: returned t
t
```

Using Wang's Algorithm on a Non-Tautology

```
(entails '() '(if A (and A B)))
 0[2]: (wang          nil  ((if A (and A B))))
   1[2]: (wang        (A)  ((and A B)))
     2[2]: (wang      (A)  (A))
     2[2]: returned t
     2[2]: (wang      (A)  (B))
     2[2]: returned nil
   1[2]: returned nil
 0[2]: returned nil
nil
```

Using Wang's Algorithm to see if

$$TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$$

```
(entails '(TD (if TD BP) (if BP (not BD))) '(not BD))
 0[2]: (wang   (TD (if TD BP) (if BP (not BD))) ((not BD)))
   1[2]: (wang (TD (if BP (not BD)))            (TD (not BD)))
   1[2]: returned t
   1[2]: (wang   (BP TD (if BP (not BD))) ((not BD)))
     2[2]: (wang (BP TD)                  (BP (not BD)))
     2[2]: returned t
     2[2]: (wang ((not BD) BP TD) ((not BD)))
     2[2]: returned t
   1[2]: returned t
 0[2]: returned t
t
```

Properties of Wang's Algorithm

1. Wang's Algorithm is **sound**:
   If `(wang A '(B)) = t` then $A \models B$
2. Wang's Algorithm is **complete**:
   If $A \models B$ then `(wang A '(B)) = t`
3. Wang's Algorithm is a **decision procedure**:
   For any valid inputs `A`, `B`,
   `(wang A '(B))` terminates
   and returns `t` iff $A \models B$

Example: Tom's Evening Domain[a]

If there is a good movie on TV and Tom doesn't have an early appointment the next morning, then he stays home and watches a late movie. If Tom needs to work and doesn't have an early appointment the next morning, then he works late. If Tom works and needs his reference materials, then he works at his office. If Tom works late at his office, then he returns to his office. If Tom watches a late movie or works late, then he stays up late.

Assume: Tom needs to work, doesn't have an early appointment, and needs his reference materials.

Prove: Tom returns to his office and stays up late.

---

[a] Based on an example in Stuart C. Shapiro, Processing, Bottom-up and Top-down, in Stuart C. Shapiro, Ed. *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, Inc., New York, 1987, 779-785.

2.3.3 Proof Theory of the Standard Propositional Logic

- Specifies when a given wfp can be derived correctly from a set of (other) wfps.

$$A_1, \ldots, A_n \vdash P$$

- Determines what wfps are **theorems** of the logic.

$$\vdash P$$

- Depends on the notion of **proof**.

Hilbert-Style Syntactic Inference

- Set of **Axioms**.
- Small set of **Rules of Inference**.

Hilbert-Style Proof

- A **proof** of a theorem P is
  - An ordered list of wfps ending with P
  - Each wfp on the list is
    - Either an axiom
    - or follows from previous wfps in the list by one of the rules of inference.

Hilbert-Style Derivation

- A **derivation** of P from $A_1, \ldots, A_n$ is
  - A list of wfps ending with P
  - Each wfp on the list is
    - Either an axiom
    - or some A_i
    - or follows from previous wfps in the list by one of the rules of inference.

Example Hilbert-Style Axioms[a]

All instances of:

(A1). $(\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A}))$

(A2). $((\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})) \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C})))$

(A3). $((\neg\mathcal{B} \Rightarrow \neg\mathcal{A}) \Rightarrow ((\neg\mathcal{B} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{B}))$

---

[a] From Elliott Mendelson, *Introduction to Mathematical Logic,* (Princeton: D. Van Nostrand) 1964, pp. 31–32.

Hilbert-Style Rule of Inference

Modus Ponens

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

Example Hilbert-Style Proof
that $\vdash A \Rightarrow A$

(1)	$(A \Rightarrow ((A \Rightarrow A) \Rightarrow A))$ $\Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))$	Instance of A2
(2)	$A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$	Instance of A1
(3)	$(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)$	From 1, 2 by MP
(4)	$A \Rightarrow (A \Rightarrow A)$	Instance of A1
(5)	$A \Rightarrow A$	From 3, 4 by MP

Example Hilbert-Style Derivation

that

Tom is the driver

Tom is the driver $\Rightarrow$ *Betty is the passenger*,

Betty is the passenger $\Rightarrow \neg$*Betty is the driver*,

$\vdash$ $\neg$*Betty is the driver*

(1)	*Tom is the driver*	Assumption
(2)	*Tom is the driver* $\Rightarrow$ *Betty is the passenger*	Assumption
(3)	*Betty is the passenger*	From 1, 2 by MP
(4)	*Betty is the passenger* $\Rightarrow \neg$*Betty is the driver*	Assumption
(5)	$\neg$*Betty is the driver*	From 3, 4 by MP

Some AI Connections

AI	Logic
domain knowledge or domain rules	assumptions or non-logical axioms
inference engine procedures	rules of inference
knowledge base	proof

Natural Deduction
(Style of Syntactic Inference)

- **No** Axioms.
- **Large** set of Rules of Inference.
  - A few **structural rules** of inference.
  - An **introduction rule** and an **elimination rule** for each connective.
- A method of **subproofs**.[a]

---

[a] Francis Jeffry Pelletier, A History of Natural Deduction and Elementary Logic Textbooks, in J. Woods, B. Brown (eds) *Logical Consequence: Rival Approaches, Vol. 1.* (Oxford: Hermes Science Pubs) 2000, pp. 105-138.

Fitch-Style Proof[a]

- A **proof** of a theorem P is
  - An ordered list of wfps and subproofs ending with P
  - Each wfp or subproof on the list must be justified by a rule of inference.
- $\vdash P$ is read "P is a theorem."

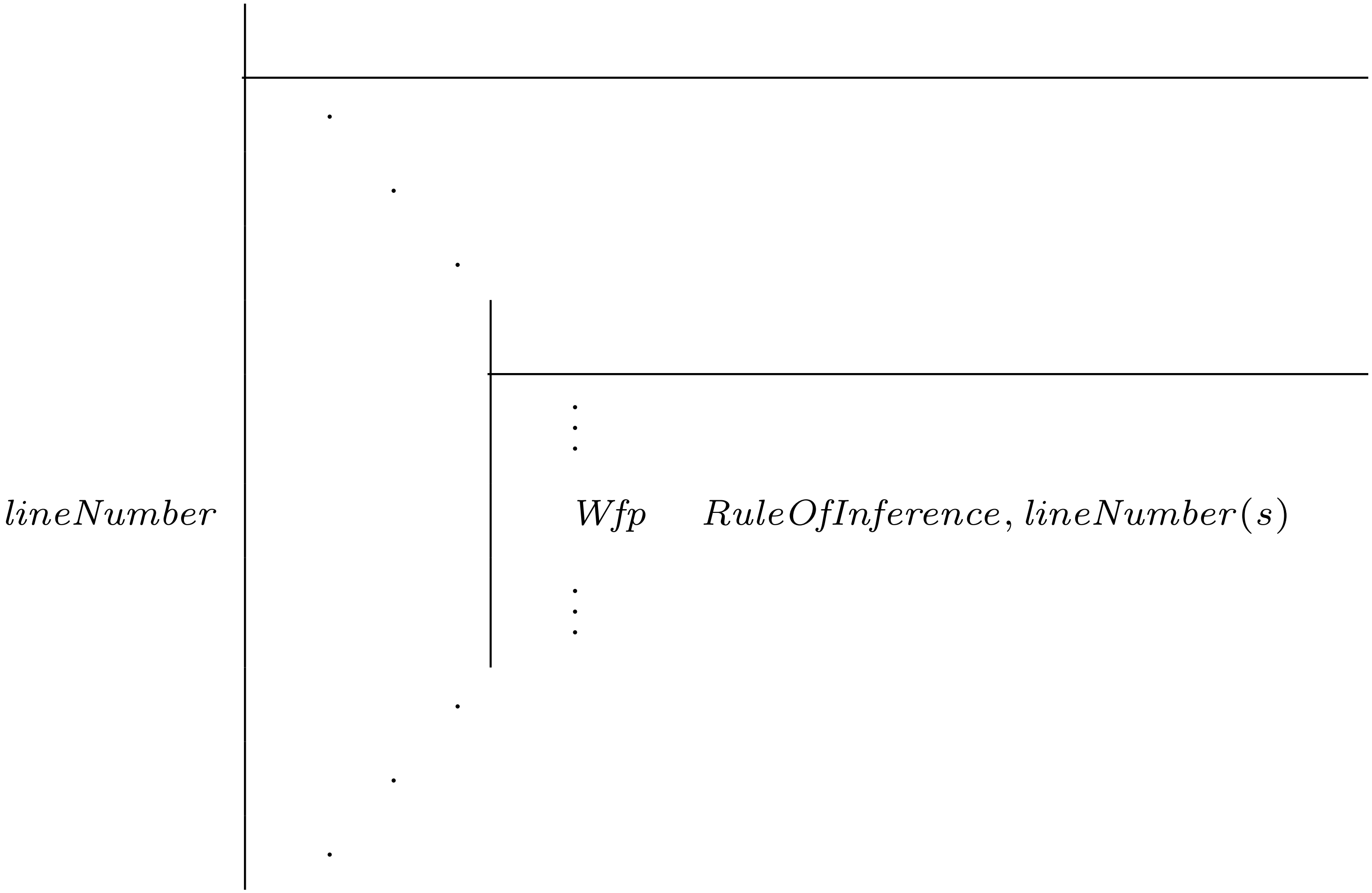
---

[a] Based on Frederic B. Fitch, *Symbolic Logic: An Introduction,* (New York: Ronald Press), 1952.

Fitch-Style Derivation

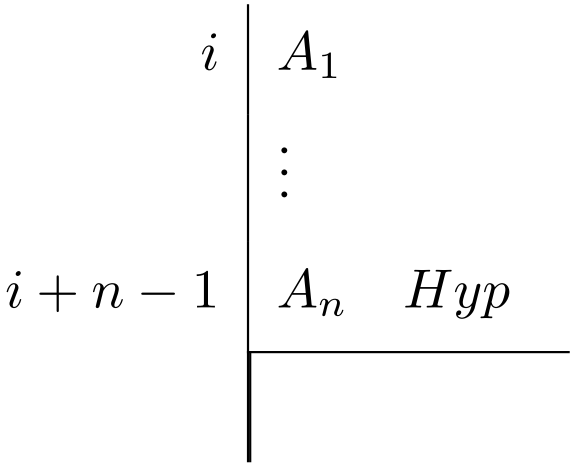
- A **derivation** of a wfp P from an assumption A is a hypothetical subproof whose hypothesis is A and which contains
  - An ordered list of wfps and inner subproofs ending with P
  - Each wfp or inner subproof on the list must be justified by a rule of inference.
- $A \vdash P$ is read "P can be derived from A."
- A Meta-theorem: $A_1 \wedge \ldots \wedge A_n \vdash P$ iff $A_1, \ldots, A_n \vdash P$

Format of Proof/Derivation



$lineNumber$ $\quad Wfp \quad RuleOfInference, lineNumber(s)$

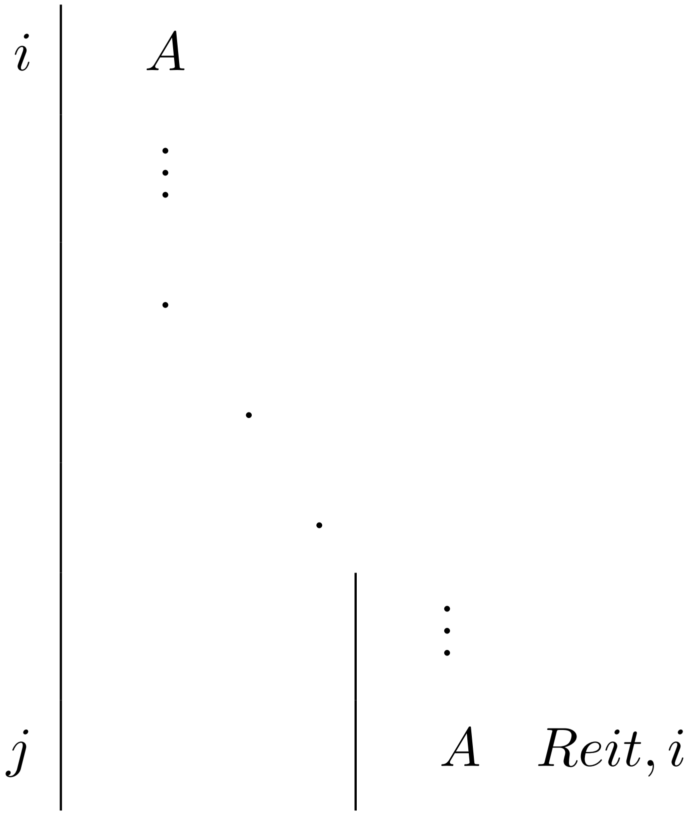
Structural Rules of Inference



i A_1

$\vdots$

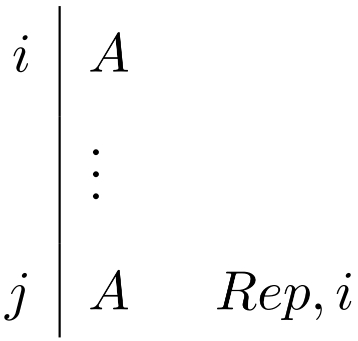
$i+n-1$ A_n Hyp



i A

$\vdots$

j A $Reit, i$



i A

$\vdots$

j A Rep, i

Rules for $\Rightarrow$

$$\begin{array}{r|l|ll}
i & & A_1 & \\
 & & \vdots & \\
i+n-1 & & A_n & Hyp \\
\cline{3-4}
 & & \vdots & \\
j & & B & \\
k & \multicolumn{2}{l}{(A_1 \wedge \ldots \wedge A_n) \Rightarrow B} & \Rightarrow I, i\text{–}j
\end{array}$$

$$\begin{array}{r|ll}
i & A & \\
 & \vdots & \\
j & A \Rightarrow B & \\
k & B & \Rightarrow E, i, j
\end{array}$$

Example Fitch-Style Proof
that $\vdash A \Rightarrow A$

$$\begin{array}{r|cl}
1. & A & Hyp \\ \cline{2-3}
2. & A & Rep, 1 \\
\multicolumn{1}{r}{3.} & A \Rightarrow A & \Rightarrow I, 1\text{–}2
\end{array}$$

Fitch-Style Proof of Axiom A1

1.	A	Hyp
2.	B	Hyp
3.	A	$Reit, 1$
4.	$B \Rightarrow A$	$\Rightarrow I, 2\text{–}3$
5.	$A \Rightarrow (B \Rightarrow A)$	$\Rightarrow I, 1\text{–}4$

Example Fitch-Style Derivation

that

$$Tom\ is\ the\ driver$$

$$Tom\ is\ the\ driver \Rightarrow Betty\ is\ the\ passenger,$$

$$Betty\ is\ the\ passenger \Rightarrow \neg Betty\ is\ the\ driver,$$

$$\vdash \qquad \neg Betty\ is\ the\ driver$$

$$\begin{array}{r|ll}
1. & Tom\ is\ the\ driver & \\
2. & Tom\ is\ the\ driver \Rightarrow Betty\ is\ the\ passenger & \\
3. & Betty\ is\ the\ passenger \Rightarrow \neg Betty\ is\ the\ driver & Hyp \\
\hline
4. & Betty\ is\ the\ passenger & \Rightarrow E, 1, 2 \\
5. & \neg Betty\ is\ the\ driver & \Rightarrow E, 4, 3
\end{array}$$

Rules for ¬

$$
\begin{array}{r|l|cl}
i. & & A_1 & \\
 & & \vdots & \\
i+n-1 & & A_n & Hyp \\
\cline{3-4}
 & & \vdots & \\
j. & & B & \\
j+1. & & \neg B & \\
j+2. & \multicolumn{3}{l}{\neg(A_1 \land \ldots \land A_n) \qquad \qquad \neg I, i\text{–}(j+1)}
\end{array}
$$

$$
\begin{array}{r|ll}
i. & \neg\neg A & \\
j. & A & \neg E, i
\end{array}
$$

Fitch-Style Proof of Axiom A3

1.	$\mid \underline{\neg B \Rightarrow \neg A}$	Hyp
2.	$\mid \mid \underline{\neg B \Rightarrow A}$	Hyp
3.	$\mid \mid \mid \underline{\neg B}$	Hyp
4.	$\mid \mid \mid \neg B \Rightarrow \neg A$	$Reit, 1$
5.	$\mid \mid \mid \neg B \Rightarrow A$	$Reit, 2$
6.	$\mid \mid \mid A$	$\Rightarrow E, 3, 5$
7.	$\mid \mid \mid \neg A$	$\Rightarrow E, 3, 4$
8.	$\mid \mid \neg\neg B$	$\neg I, 3–7$
9.	$\mid \mid B$	$\neg E, 8$
10.	$\mid (\neg B \Rightarrow A) \Rightarrow B$	$\Rightarrow I, 2–9$
11.	$(\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B)$	$\Rightarrow I, 1–10$

Rules for $\wedge$

$$\begin{array}{r|ll}
i_1. & A_1 & \\
 & \vdots & \\
i_n. & A_n & \\
j. & A_1 \wedge \cdots \wedge A_n & \wedge I, i_1, \ldots, i_n
\end{array}$$

$$\begin{array}{r|ll}
i. & A_1 \wedge \cdots \wedge A_n & \\
 & \vdots & \\
j. & A_k & \wedge E, i
\end{array}$$

Proof that
$\vdash (A \wedge B \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))$

1.	$A \wedge B \Rightarrow C$	Hyp
2.	A	Hyp
3.	B	Hyp
4.	A	$Reit, 2$
5.	$A \wedge B$	$\wedge I, 4, 3$
6.	$A \wedge B \Rightarrow C$	$Reit, 1$
7.	C	$\Rightarrow E, 5, 6$
8.	$B \Rightarrow C$	$\Rightarrow I, 3–7$
9.	$A \Rightarrow (B \Rightarrow C)$	$\Rightarrow I, 2–8$
10.	$(A \wedge B \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))$	$\Rightarrow I, 1–9$

Proof that $\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \wedge B \Rightarrow C)$

1.	$A \Rightarrow (B \Rightarrow C)$	Hyp
2.	$A \wedge B$	Hyp
3.	A	$\wedge E, 2$
4.	B	$\wedge E, 2$
5.	$A \Rightarrow (B \Rightarrow C)$	$Reit, 1$
6.	$B \Rightarrow C$	$\Rightarrow E, 3, 5$
7.	C	$\Rightarrow E, 4, 6$
8.	$A \wedge B \Rightarrow C$	$\Rightarrow I, 2–7$
9.	$(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \wedge B \Rightarrow C)$	$\Rightarrow I, 1–8$

Rules for $\vee$

$$\begin{array}{r|ll} i. & A_i & \\ j. & A_1 \vee \cdots \vee A_i \vee \cdots \vee A_n & \vee I, i \end{array}$$

$$\begin{array}{r|ll} i. & A_1 \vee \cdots \vee A_n & \\ & \vdots & \\ j_1. & \quad A_1 \Rightarrow B & \\ & \vdots & \\ j_n. & \quad A_n \Rightarrow B & \\ k. & \qquad B & \vee E, i, j_1, \ldots, j_n \end{array}$$

Proof that

$\vdash (A \Rightarrow B) \Rightarrow (\neg A \vee B)$

1.	$A \Rightarrow B$	Hyp
2.	$\neg(\neg A \vee B)$	Hyp
3.	$\neg A$	Hyp
4.	$\neg A \vee B$	$\vee I, 3$
5.	$\neg(\neg A \vee B)$	$Reit, 2$
6.	$\neg\neg A$	$\neg I, 3$–5
7.	A	$\neg E, 6$
8.	$A \Rightarrow B$	$Reit, 1$
9.	B	$\Rightarrow E, 7, 8$
10.	$\neg A \vee B$	$\vee I, 9$
11.	$\neg(\neg A \vee B)$	$Rep, 2$
12.	$\neg\neg(\neg A \vee B)$	$\neg I, 2$–11
13.	$\neg A \vee B$	$\neg E, 12$
14.	$(A \Rightarrow B) \Rightarrow (\neg A \vee B)$	$\Rightarrow I, 1$–14

Proof that $\vdash (A \vee B) \wedge \neg A \Rightarrow B$

1.	$(A \vee B) \wedge \neg A$	Hyp
2.	$\neg A$	$\wedge E$, 1
3.	$A \vee B$	$\wedge E$, 1
4.	A	Hyp
5.	$\neg B$	Hyp
6.	A	$Reit$, 4
7.	$\neg A$	$Reit$, 2
8.	$\neg\neg B$	$\neg I$, 5–7
9.	B	$\neg E$, 8
10.	$A \Rightarrow B$	$\Rightarrow I$, 4–9
11.	B	Hyp
12.	B	Rep, 11
13.	$B \Rightarrow B$	$\Rightarrow I$, 11–12
14.	B	$\vee E$, 3, 10, 13
15.	$(A \vee B) \wedge \neg A \Rightarrow B$	$\Rightarrow I$, 1–14

Rules for $\Leftrightarrow$

$$\begin{array}{r|ll} i. & A \Rightarrow B & \\ & \vdots & \\ j. & B \Rightarrow A & \\ k. & A \Leftrightarrow B & \quad \Leftrightarrow I, i, j \end{array}$$

$$\begin{array}{r|ll} i. & A & \\ & \vdots & \\ j. & A \Leftrightarrow B & \\ k. & B & \qquad \Leftrightarrow E, i, j \end{array}$$

$$\begin{array}{r|ll} i. & B & \\ & \vdots & \\ j. & A \Leftrightarrow B & \\ k. & A & \qquad \Leftrightarrow E, i, j \end{array}$$

Proof that

$\vdash (A \Rightarrow (B \Rightarrow C)) \Leftrightarrow (A \wedge B \Rightarrow C)$

$\boxed{\text{Proof from p. 120}}$

9. $(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \wedge B \Rightarrow C)$ $\quad \Rightarrow I$

$\boxed{\text{Proof from p. 119}}$

18. $(A \wedge B \Rightarrow C) \Rightarrow (A \Rightarrow (B \Rightarrow C))$ $\quad \Rightarrow I$

19. $(A \Rightarrow (B \Rightarrow C)) \Leftrightarrow (A \wedge B \Rightarrow C)$ $\quad \Leftrightarrow I, 9, 18$

CarPool World Derivation

1.	$Tom\ is\ the\ driver \Leftrightarrow \neg Tom\ is\ the\ passenger$	
2.	$Tom\ is\ the\ passenger \Leftrightarrow Betty\ is\ the\ driver$	
3.	$Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger$	
4.	$Tom\ is\ the\ driver$	Hyp
5.	$\neg Tom\ is\ the\ passenger$	$\Leftrightarrow E, 4, 1$
6.	$\neg Betty\ is\ the\ passenger$	Hyp
7.	$Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger$	$Reit, 3$
8.	$Betty\ is\ the\ driver$	$\Leftrightarrow E, 6, 7$
9.	$Tom\ is\ the\ passenger \Leftrightarrow Betty\ is\ the\ driver$	$Reit, 2$
10.	$Tom\ is\ the\ passenger$	$\Leftrightarrow E, 8, 9$
11.	$\neg Tom\ is\ the\ passenger$	$Reit, 5$
12.	$\neg\neg Betty\ is\ the\ passenger$	$\neg I, 6–11$
13.	$Betty\ is\ the\ passenger$	$\neg E, 12$

Implementing Natural Deduction

Heuristics:

If trying to prove/derive a non-atomic wfp,
try the introduction rule for the major connective.

If trying to prove/derive a wfp,
and that wfp is a component of a wfp,
try the relevant elimination rule.

Using SNePS 3

```
cl-user(2): :ld /projects/snwiz/Sneps3/sneps3
...
"Change package to snuser."
cl-user(3): :pa snuser

snuser(4): (showproofs)
nil

snuser(5): (askif '(if A A))
Let me assume that A
Since A can be derived after assuming A
I infer wft1!: (if A A) by Implication Introduction.
wft1!: (if A A)
```

Derivation by SNePS 3

```
snuser(12): (clearkb)
nil

snuser(13): (assert ’TomIsTheDriver)
TomIsTheDriver!

snuser(14): (assert ’(if TomIsTheDriver BettyIsThePassenger))
wft1!: (if TomIsTheDriver! BettyIsThePassenger)

snuser(15): (assert ’(if BettyIsThePassenger (not BettyIsTheDriver)))
wft3!: (if BettyIsThePassenger (not BettyIsTheDriver))

snuser(16): (askif ’(not BettyIsTheDriver))
Since wft1!: (if TomIsTheDriver! BettyIsThePassenger)
 and TomIsTheDriver!
I infer BettyIsThePassenger! by Implication Elimination.

Since wft3!: (if BettyIsThePassenger! (not BettyIsTheDriver))
 and BettyIsThePassenger!
I infer wft2!: (not BettyIsTheDriver) by Implication Elimination.

wft2!: (not BettyIsTheDriver)

snuser(17): (askif ’BettyIsThePassenger) ; Lemma
BettyIsThePassenger!
```

SNePS 3 Proves Axiom A1

```
snuser(9): (clearkb)
nil

snuser(10): (askif '(if A (if B A)))
Let me assume that A
Let me assume that B
Since A can be derived after assuming B
I infer wft1?: (if B A) by Implication Introduction.

Since wft1?: (if B A) can be derived after assuming A
I infer wft2!: (if A (if B A)) by Implication Introduction.
wft2!: (if A (if B A))
```

Another Derivation by SNePS 3

```
snuser(24): (clearkb)
nil
snuser(25): (assert '(iff TomIsTheDriver (not TomIsThePassenger)))
wft2!: (iff TomIsTheDriver (not TomIsThePassenger))
snuser(26): (assert '(iff TomIsThePassenger BettyIsTheDriver))
wft3!: (iff TomIsThePassenger BettyIsTheDriver)
snuser(27): (assert '(iff BettyIsTheDriver (not BettyIsThePassenger)))
wft5!: (iff (not BettyIsThePassenger) BettyIsTheDriver)
snuser(28): (assert 'TomIsTheDriver)
TomIsTheDriver!

snuser(29): (askif 'BettyIsThePassenger)
Since wft2!: (iff TomIsTheDriver! (not TomIsThePassenger))
 and TomIsTheDriver!
I infer wft1!: (not TomIsThePassenger) by Equivalence Elimination.

Since wft3!: (iff TomIsThePassenger BettyIsTheDriver)
 and wft1!: (not TomIsThePassenger)
I infer wft7!: (not BettyIsTheDriver) by Equivalence Elimination.

Since wft5!: (iff (not BettyIsThePassenger) BettyIsTheDriver)
 and wft7!: (not BettyIsTheDriver)
I infer BettyIsThePassenger! by Equivalence Elimination.

BettyIsThePassenger!
```

More Connections

- Deduction Theorem: $A \vdash P$ if and only if $\vdash A \Rightarrow P$.
- So proving theorems and deriving conclusions from assumptions are equivalent.
- But no atomic proposition is a theorem.
- So theorem proving makes more use of Introduction Rules than most AI reasoning systems.

2.4 Important Properties of Logical Systems

Soundness: $\vdash P$ implies $\models P$

Consistency: not both $\vdash P$ and $\vdash \neg P$

Completeness: $\models P$ implies $\vdash P$

More Connections

- If at most 1 of $\models P$ and $\models \neg P$ then soundness implies consistency.
- Soundness is the essence of "correct reasoning."
- Completeness less important for running systems since a proof may take too long to wait for.
- The Propositional Logics we have been looking at are complete.
- Gödel's Incompleteness Theorem: A logic strong enough to formalize arithmetic is *either* inconsistent *or* incomplete.

More Connections

$$\begin{array}{ccc}
A_1, \ldots, A_n \vdash P & \Leftrightarrow & \vdash A_1 \wedge \ldots \wedge A_n \Rightarrow P \\[2em]
\text{soundness} \Downarrow\!\Uparrow \text{completeness} & & \text{soundness} \Downarrow\!\Uparrow \text{completeness} \\[2em]
A_1, \ldots, A_n \models P & \Leftrightarrow & \models A_1 \wedge \ldots \wedge A_n \Rightarrow P
\end{array}$$

2.5 Clause Form Propositional Logic

2.5.1 Clause Form Syntax

Syntax of Atoms and Literals

Atomic Propositions:

- Any letter of the alphabet
- Any letter with a numerical 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**.

Clause Form Syntax
Syntax of Clauses and Sets of Clauses

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.

2.5.2 Clause Form Semantics

Atomic Propositions

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

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

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.

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.

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.

2.5.3 Clause Form Proof Theory: Resolution

Notion of Proof: None!

Notion of Derivation: A set of clauses constitutes a derivation.

Assumptions: The derivation is initialized with a set of assumption clauses $A_1, \ldots, A_n$.

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

Derived Clause: If clause Q has been added to a set of clauses initialized with the set of assumption clauses $A_1, \ldots, A_n$ by one or more applications of resolution,
then $A_1, \ldots, A_n \vdash Q$.

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

$\{P\} \not\vdash \{P, Q\}$

because resolution does not apply to $\{\{P\}\}$.

2.5.4 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

- Set of clauses $\{A_1, \ldots, A_n, Q\}$ is contradictory
- means $(A_1 \wedge \ldots \wedge A_n \wedge Q)$ is False in all models
- means whenever $(A_1 \wedge \ldots \wedge A_n)$ is True, Q is False
- means whenever $(A_1 \wedge \ldots \wedge A_n)$ is True $\neg Q$ is True
- means $A_1, \ldots, A_n \models \neg Q$.

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 $A_1, \ldots, A_n \models Q$:

1. Let $S = \{A_1, \ldots, A_n\} \cup \neg Q$ (Note: $\neg Q$ is a set of clauses.)
2. Repeatedly apply resolution to clauses in S.
   (Determine if $\{A_1, \ldots, A_n\} \cup \neg Q \vdash \{\}$)
3. If generate $\{\}$, $A_1, \ldots, A_n \models Q$.
   (If $\{A_1, \ldots, A_n\} \cup \neg Q \vdash \{\}$ then $A_1, \ldots, A_n \models Q$)
4. If reach point where no new clause can be generated,
   but $\{\}$ has not appeared, $A_1, \ldots, A_n \not\models Q$.
   (If $\{A_1, \ldots, A_n\} \cup \neg Q \not\vdash \{\}$ then $A_1, \ldots, A_n \not\models Q$)

Example 1

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

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

1.	$\{P\}$	$Assumption$
2.	$\{\neg P\}$	$From\ query\ clause$
3.	$\{\}$	$R, 1, 2$

Example 2

To decide if

$$\begin{aligned}&\{\neg TomIsTheDriver, \neg TomIsThePassenger\},\\ &\{TomIsThePassenger, BettyIsThePassenger\},\\ &\{TomIsTheDriver\} \quad \models \{BettyIsThePassenger\}\end{aligned}$$

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. (Check throughout.)

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. (Check throughout.)

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. (Check throughout.)

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

```
cl-user(2): :ld /projects/shapiro/AIclass/prover
; Fast loading /projects/shapiro/AIclass/prover.fasl

cl-user(3): :pa prover

prover(4): (prove '(P) '(or P Q))
 1  (P)               Assumption
 2  ((not P))         From Query
 3  ((not Q))         From Query
Deleting  3 ((not Q))
 because Q is not used positively in any clause.
 4  nil               R,2,1,{}
QED
```

Example 2 Using prover

```
prover(19): (prove '((or (not TomIsTheDriver) (not TomIsThePassenger))
                     (or TomIsThePassenger BettyIsThePassenger)
                     TomIsTheDriver)
                   'BettyIsThePassenger)
 1  (TomIsTheDriver)  Assumption
 2  ((not TomIsTheDriver) (not TomIsThePassenger))  Assumption
 3  (TomIsThePassenger BettyIsThePassenger) Assumption
 4  ((not BettyIsThePassenger)) From Query
 5  (TomIsThePassenger) R,4,3,{}
Deleting  3 (TomIsThePassenger BettyIsThePassenger)
because it's subsumed by  5 (TomIsThePassenger)
 6  ((not TomIsTheDriver))  R,5,2,{}
Deleting  2 ((not TomIsTheDriver) (not TomIsThePassenger))
because it's subsumed by  6 ((not TomIsTheDriver))
 7  nil             R,6,1,{}
QED
```

Example 1 Using SNARK

```
cl-user(5): :ld /projects/shapiro/CSE563/snark
; Loading /projects/shapiro/CSE563/snark.cl
...
cl-user(6): :pa snark-user
snark-user(7): (initialize)
...

snark-user(8): (assert 'P)
nil

snark-user(9): (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 Elimination and the Unit-Preference Strategy are included.

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

2.5.5 Equivalence of Standard Propositional Logic and Clause FormLogic

Every set of clauses,

$$\{\{L_{1,1}, \ldots, L_{1,n_1}\}, \ldots, \{L_{m,1}, \ldots, L_{m,n_m}\}\}$$

has the same semantics as the standard wfp

$$((L_{1,1} \vee \cdots \vee L_{1,n_1}) \wedge \cdots \wedge (L_{m,1} \vee \cdots \vee L_{m,n_m}))$$

That is, there is a translation from any set of clauses into a well-formed proposition of standard propositional logic.

Question: Is there a translation from any well-formed proposition of standard propositional logic into a set of clauses?

Answer: Yes!

Translating Standard Wfps into Clause Form Conjunctive Normal Form (CNF)

A standard wfp is in **CNF** if it is a conjunction of disjunctions of literals.

$$((L_{1,1} \vee \cdots \vee L_{1,n_1}) \wedge \cdots \wedge (L_{m,1} \vee \cdots \vee L_{m,n_m}))$$

Translation technique:

1. Turn any arbitrary wfp into CNF.
2. Translate the CNF wfp into a set of clauses.

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

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

From: $(\mathit{LivingThing} \Leftrightarrow (\mathit{Animal} \lor \mathit{Vegetable}))$

To:

$((\mathit{LivingThing} \Rightarrow (\mathit{Animal} \lor \mathit{Vegetable}))$

$\land((\mathit{Animal} \lor \mathit{Vegetable}) \Rightarrow \mathit{LivingThing}))$

Translation Step 2

Eliminate occurrences of $\Rightarrow$ using

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

From:

$((\mathit{LivingThing} \Rightarrow (\mathit{Animal} \vee \mathit{Vegetable}))$
$\wedge((\mathit{Animal} \vee \mathit{Vegetable}) \Rightarrow \mathit{LivingThing}))$

To:

$((\neg \mathit{LivingThing} \vee (\mathit{Animal} \vee \mathit{Vegetable}))$
$\wedge(\neg(\mathit{Animal} \vee \mathit{Vegetable}) \vee \mathit{LivingThing}))$

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:

$((\neg \mathit{LivingThing} \vee (\mathit{Animal} \vee \mathit{Vegetable}))$
$\wedge(\neg(\mathit{Animal} \vee \mathit{Vegetable}) \vee \mathit{LivingThing}))$

To:

$((\neg \mathit{LivingThing} \vee (\mathit{Animal} \vee \mathit{Vegetable}))$
$\wedge((\neg \mathit{Animal} \wedge \neg \mathit{Vegetable}) \vee \mathit{LivingThing}))$

Translation Step 4

CNF: Translate into Conjunctive Normal Form, using

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

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

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 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:

$((\neg \mathit{LivingThing} \vee \mathit{Animal} \vee \mathit{Vegetable})$
$\wedge(\neg \mathit{Animal} \vee \mathit{LivingThing})$
$\wedge(\neg \mathit{Vegetable} \vee \mathit{LivingThing}))$

To:

$\{\{\neg \mathit{LivingThing},\ \mathit{Animal},\ \mathit{Vegetable}\},$
$\{\neg \mathit{Animal},\ \mathit{LivingThing}\},$
$\{\neg \mathit{Vegetable},\ \mathit{LivingThing}\}\}$

Use of Translation

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

iff

The translation of $A_1 \wedge \cdots \wedge A_n \wedge \neg Q$ into a set of clauses

$$\vdash \{\}$$

`prover` Example

To prove

$(LivingThing \Leftrightarrow Animal \vee Vegetable), (LivingThing \wedge \neg Animal) \models Vegetable$

```
prover(20): (prove '((iff LivingThing (or Animal Vegetable))
                     (and LivingThing (not Animal)))
                   'Vegetable)
 1  (LivingThing)   Assumption
 2  ((not Animal))  Assumption
 3  ((not Animal) LivingThing)  Assumption
 4  ((not Vegetable) LivingThing) Assumption
 5  ((not LivingThing) Animal Vegetable)  Assumption
 6  ((not Vegetable)) From Query
Deleting  3 ((not Animal) LivingThing)
because it's subsumed by  1 (LivingThing)
Deleting  4 ((not Vegetable) LivingThing)
because it's subsumed by  1 (LivingThing)
```

prover Example, continued

```
 1  (LivingThing)   Assumption
 2  ((not Animal))  Assumption
 5  ((not LivingThing) Animal Vegetable)  Assumption
 6  ((not Vegetable)) From Query

 7  ((not LivingThing) Animal)  R,6,5,{}
Deleting  5 ((not LivingThing) Animal Vegetable)
because it's subsumed by  7 ((not LivingThing) Animal)
 8  (Animal)        R,7,1,{}
 9  ((not LivingThing)) R,7,2,{}
10  nil             R,9,1,{}
QED
```

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**
$\{\neg A_1, \ldots, \neg A_n, C\}$	$(A_1 \wedge \cdots \wedge A_n) \Rightarrow C$
Set of Support	**Back-chaining**

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

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

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

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 $\forall x P$ and $\exists x P$.

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

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

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

$[Tom]$ = Someone named Tom.
$[Betty]$ = 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

$[\![Tom]\!] =$ Tom.

$[\![Betty]\!] =$ Betty.

$[\![John]\!] =$ John.

$[\![Mary]\!] =$ Mary.

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

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

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

4 Full First-Order Predicate Logic (FOL)

4.1 CarPool World

We'll add Tom and Betty's mothers:

$motherOf(Tom)$ and $motherOf(Betty)$

CarPool World Domain Rules (Partial)

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

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

4.2 The “Standard” First-Order Predicate Logic

4.2.1 Syntax of the "Standard" First-Order 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

Arbitrary Individuals:

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

Indefinite Individuals:

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

Atomic Symbols, Part 3

Variables:

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

For example: x, y_6.

Atomic Symbols, Part 4

Function Symbols:

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

For example: $f, g_2, motherOf, familyOf.$

Atomic Symbols, Part 5

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: $P, Q_4, Passenger, Drives$.

Atomic Symbols, Part 6

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

Use superscript for explicit arity.

For example: $motherOf^1, Drives^2, familyOf^2, g_2^3$

Atomic Symbols, Part 7

In any specific predicate logic language

Individual Constants,
Arbitrary Individuals,
Indefinite Individuals,
Variables,
Function Symbols,
Predicate Symbols

must be disjoint.

Terms

- Every individual constant, every arbitrary individual, every indefinite individual, and every variable is a term.
- If f^n is a function symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $f^n(t_1, \ldots, t_n)$ is a term.
  (The superscript may be omitted if no confusion results.)
  For example: $\mathit{familyOf}^{\mathit{2}}(\mathit{Tom}, \mathit{motherOf}^{\mathit{1}}(\mathit{Betty}))$
- 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.: $\mathit{ChildIn}^2(\mathit{Betty}, \mathit{familyOf}^2(\mathit{Tom}, \mathit{motherOf}^1(\mathit{Betty})))$

(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

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

  $(P \Rightarrow Q) \quad (P \Leftrightarrow Q)$
- 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 $\exists x, y(P)$.

Open, Closed, Ground, and Free For

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**.

Note: expressions now include functional terms.

A term t is **free for** a variable x in the wff $A(x)$ if
no free occurrence of x in $A(x)$ is in the scope
of any quantifier $\forall y$ or $\exists y$ whose variable y is in t.
E.g., $f(a, y, b)$ is free for x in $\forall u \exists v(A(x, u) \vee B(x, v))$
but $f(a, y, b)$ is not free for x in $\forall u \exists y(A(x, u) \vee B(x, y))$.
Remedy: rename y in $A(x)$. E.g., $\forall u \exists v(A(x, u) \vee B(x, v))$

Substitutions with Functional Terms

Notice, terms may now include functional terms.

E.g.:

$$P(x, f(y), z)\{a/x, g(b)/y, f(a)/z\} = P(a, f(g(b)), f(a))$$

4.2.2 Semantics of the "Standard" First-Order Predicate Logic

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

- individuals,
- functions on individuals,
- sets of individuals,
- relations on individuals

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

Semantics of Constants

Individual Constant:

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

Arbitrary Individual:

$[a] = [\![a]\!]$ = a representative of all individuals in $\mathcal{I}$. Everything True about all of them, is True of it.

Indefinite Individual:

$[s] = [\![s]\!]$ = a representative of some individual in $\mathcal{I}$, but it's unspecified which one.

There is no anonymous individual.
I.e. for every individual, i in $\mathcal{I}$, there is a ground term t such that $[\![t]\!]$ = i. (But not necessarily an individual constant.)

Intensional Semantics of Functional Terms

Function Symbols: $[f^n]$ is some n-ary function in $\mathcal{D}$,

Functional Terms:

If f^n is some function symbol and $t_1, \ldots, t_n$ are ground terms, then $[f^n(t_1, \ldots, t_n)]$ is a description of the individual in $\mathcal{I}$ that is the value of $[f^n]$ on $[t_1]$, and $\ldots$, and $[t_n]$.

Extensional Semantics of Functional Terms

Function Symbols: $[\![f^n]\!]$ is some function in $\mathcal{D}$,

$$[\![f^n]\!]\colon \underbrace{\mathcal{I} \times \cdots \times \mathcal{I}}_{n \text{ times}} \to \mathcal{I}$$

Functional Terms:

If f^n is some function symbol and $t_1, \ldots, t_n$ are ground terms, then $[\![f^n(t_1, \ldots, t_n)]\!] = [\![f^n]\!]([\![t_1]\!], \ldots, [\![t_n]\!])$.

Semantics of Predicate Symbols

Predicate Symbols:

- $[P^{1}]$ is some category/property of individuals of $\mathcal{I}$
- $[P^{n}]$ is some n-ary relation in $\mathcal{D}$.
- $[\![P^{1}]\!]$ 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 ground term,
  then $[P^{\mathit{1}}(t)]$ is the proposition that $[t]$ is an instance of the category $[P^{\mathit{1}}]$ (or has the property $[P^{\mathit{1}}]$).
- If P^n is some n-ary predicate symbol,
  and $t_1, \ldots, t_n$ are ground terms,
  then $[P^n(t_{\mathit{1}}, \ldots, t_n)]$ is the proposition that the relation $[P^n]$ holds among individuals $[t_{\mathit{1}}]$, and …, and $[t_n]$.

Extensional Semantics of Ground Atomic Formulas

Atomic Formulas:

- If P^1 is some unary predicate symbol,
  and t is some ground term,
  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 ground terms,
  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

$[\neg P]$, $[P \land Q]$, $[P \lor Q]$, $[P \Rightarrow Q]$, $[P \Leftrightarrow Q]$
$[\![\neg P]\!]$, $[\![P \land Q]\!]$, $[\![P \lor Q]\!]$, $[\![P \Rightarrow Q]\!]$, 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 or description t_i, satisfies $[P\{t_i/x\}]$.
- $[\exists x P]$ is the proposition that some individual i in $\mathcal{I}$, with name or description t_i, satisfies $[P\{t_i/x\}]$.
- $[\![\forall x P]\!]$ is True if $[\![P\{t/x\}]\!]$ is True for every ground term, t. Otherwise, it is False.
- $[\![\exists x P]\!]$ is True if there is some ground term, t such that $[\![P\{t/x\}]\!]$ is True. Otherwise, it is False.

Intensional Semantics
of a 2-Car CarPool World 1

Individual Constants:

$[Tom]$ = The individual named Tom.

$[Betty]$ = The individual named Betty.

Functions:

$[motherOf(x)]$ = The mother of $[x]$.

Intensional Semantics of a 2-Car CarPool World 2

Predicates:

$[\mathit{Driver}^{\mathit{1}}(x)] = [x]$ is the driver of a car.

$[\mathit{Passenger}^{\mathit{1}}(x)] = [x]$ is the passenger in a car.

$[\mathit{Drives}^{\mathit{2}}(x, y)] = [x]$ drives $[y]$ in a car.

Extensional Semantics of a 2-Car CarPool World Situation

$[\![Tom]\!]$ = the individual named Tom.

$[\![Betty]\!]$ = the individual named Betty.

$$[\![motherOf]\!] = \{\langle\ [\![Betty]\!], [\![motherOf(Betty)]\!]\ \rangle,\\ \langle\ [\![Tom]\!], [\![motherOf(Tom)]\!]\rangle\}.$$

$$[\![Driver]\!] = \{[\![motherOf(Betty)]\!], [\![motherOf(Tom)]\!]\}.$$

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

$$[\![Drives]\!] = \{\langle\ [\![motherOf(Betty)]\!], [\![Betty]\!]\ \rangle,\\ \langle\ [\![motherOf(Tom)]\!], [\![Tom]\!]\ \rangle\}.$$

4.2.3 Model Checking in Full FOL

n Individual Constants.

At least one function yields ∞ terms.$^{*Decreasoner.}$
E.g., $motherOf(Tom), motherOf(motherOf(Tom)),$
$motherOf(motherOf(motherOf(Tom)))\ldots.$

So ∞ ground atomic propositions.

So ∞ situations (columns of truth table).

So can't create entire truth table.

Can't do model checking
by expanding quantified expressions
into Boolean combination of ground wffs.

There still could be a finite domain if at least one individual in $\mathcal{I}$ has an ∞ number of terms describing it, but we'll assume not.

4.2.4 Hilbert-Style Proof Theory for First-Order Predicate Logic

(A1). $(\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{A}))$

(A2). $((\mathcal{A} \Rightarrow (\mathcal{B} \Rightarrow \mathcal{C})) \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{C})))$

(A3). $((\neg\mathcal{B} \Rightarrow \neg\mathcal{A}) \Rightarrow ((\neg\mathcal{B} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{B}))$

(A4). $\forall x\mathcal{A} \Rightarrow \mathcal{A}\{t/x\}$
where t is any term free for x in $A(x)$.

(A5). $(\forall x(\mathcal{A} \Rightarrow \mathcal{B})) \Rightarrow (\mathcal{A} \Rightarrow \forall x\mathcal{B})$
if $\mathcal{A}$ is a wff containing no free occurrences of x.

Hilbert-Style Rules of Inference for "Standard" First-Order Predicate Logic

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

$$\frac{\mathcal{A}}{\forall x \mathcal{A}}$$

Note: $\exists x \mathcal{A}$ is just an abbreviation of $\neg \forall x \neg \mathcal{A}$.

4.2.5 Fitch-Style Proof Theory for First-Order Predicate Logic

Additional Rules of Inference for ∀

$$\begin{array}{r|ll}
i & \underline{\;a\quad} & \text{Arb I} \\
 & \vdots & \\
j & P(a) & \\
\multicolumn{1}{r}{j+1} & \multicolumn{1}{l}{\forall x P\{x/a\}} & \forall I, i\text{–}j
\end{array}$$

$$\begin{array}{r|ll}
i & \forall x P(x) & \\
i+1 & P\{t/x\} & \forall E, i
\end{array}$$

Where a is an arbitrary individual not otherwise used in the proof, and t is any term, whether or not used elsewhere in the proof, that is free for x in $P(x)$.

Example of ∀ Rules

To prove $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$

$$\begin{array}{lll}
1 & \vert\ \underline{\forall x(P(x) \Rightarrow Q(x))} & \text{Hyp} \\
2 & \vert\ \vert\ \underline{\forall x P(x)} & \text{Hyp} \\
3 & \vert\ \vert\ \vert\ \underline{a} & \text{Arb I} \\
4 & \vert\ \vert\ \vert\ \forall x P(x) & \text{Reit, 2} \\
5 & \vert\ \vert\ \vert\ P(a) & \forall\text{E, 4} \\
6 & \vert\ \vert\ \vert\ \forall x(P(x) \Rightarrow Q(x)) & \text{Reit, 1} \\
7 & \vert\ \vert\ \vert\ P(a) \Rightarrow Q(a) & \forall\text{E, 6} \\
8 & \vert\ \vert\ \vert\ Q(a) & \Rightarrow\text{E, 5,7} \\
9 & \vert\ \vert\ \forall x Q(x) & \forall\text{I, 3–8} \\
10 & \vert\ \forall x P(x) \Rightarrow \forall x Q(x) & \Rightarrow\text{I, 2–9} \\
11 & \forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x)) & \Rightarrow\text{I, 1–10}
\end{array}$$

Additional Rules of Inference for ∃

$$\begin{array}{r|ll} i & P(t) & \\ i+1 & \exists x P(x) & \exists I, i \end{array}$$

$$\begin{array}{r|c|ll}
i & \multicolumn{1}{c}{} & \exists x P(x) & \\
 & \multicolumn{1}{c}{} & \vdots & \\
j & & P\{a/x\} & \text{Indef I}, i \\ \cline{3-4}
 & & \vdots & \\
k & & Q & \\
k+1 & \multicolumn{1}{c}{} & Q & \exists E, j\text{–}k
\end{array}$$

Where $P(x)$ is the result of replacing some or all occurrences of t in $P(t)$ by x,

t is free for x in $P(x)$;

a is an indefinite individual not otherwise used in the proof,

$P(a/x)$ is the result of replacing all occurrences of x in $P(x)$ by a,

and there is no occurrence of a in Q. (Compare $\exists E$ to $\vee E$.)

Example of ∃ Rules

To prove $\exists x(P(x) \wedge Q(x)) \Rightarrow (\exists x P(x) \wedge \exists x Q(x))$

1	$\quad \exists x(P(x) \wedge Q(x))$	Hyp
2	$\quad\quad P(a) \wedge Q(a)$	Indef I, 1
3	$\quad\quad P(a)$	∧E, 2
4	$\quad\quad \exists x P(x)$	∃I, 3
5	$\quad \exists x P(x)$	∃E, 2–4
6	$\quad\quad P(b) \wedge Q(b)$	Indef I, 1
7	$\quad\quad Q(b)$	∧E, 5
8	$\quad\quad \exists x Q(x)$	∃I, 6
9	$\quad \exists x Q(x)$	∃E, 5–7
10	$\quad \exists x P(x) \wedge \exists x Q(x)$	∧I, 5,9
11	$\exists x(P(x) \wedge Q(x)) \Rightarrow (\exists x P(x) \wedge \exists x Q(x))$	⇒I, 1–10

CarPool Situation Derivation

1	$\forall x(Driver(x) \Rightarrow \neg Passenger(x))$	
2	$\forall x \forall y(Drives(x,y) \Rightarrow (Driver(x) \wedge Passenger(y)))$	
3	$\forall x Drives(motherOf(x), x)$	Hyp
4	$Drives(motherOf(Tom), Tom)$	$\forall E, 3$
5	$\forall y(Drives(motherOf(Tom), y)$ $\Rightarrow (Driver(motherOf(Tom)) \wedge Passenger(y)))$	$\forall E, 2$
6	$Drives(motherOf(Tom), Tom)$ $\Rightarrow (Driver(motherOf(Tom)) \wedge Passenger(Tom))$	$\forall E, 5$
7	$Driver(motherOf(Tom)) \wedge Passenger(Tom)$	$\Rightarrow E, 4, 6$
8	$Driver(motherOf(Tom))$	$\wedge E, 7$
9	$\exists x Driver(motherOf(x))$	$\exists I, 8$

4.3 Clause-Form First-Order Predicate Logic

4.3.1 Syntax of Clause-Form First-Order 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*.

Skolem Constants: Look like individual constants.

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

Function Symbols:

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

For example: f, g_2.

Use superscript for explicit arity.

Skolem Function Symbols: Look like function symbols.

Atomic Symbols, Part 4

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, odd.

Use superscript for explicit arity.

Terms

- Every individual constant, every Skolem constant, and every variable is a term.
- If f^n is a function symbol or Skolem function symbol of arity n, and $t_1, \ldots, t_n$ are terms,
  then $f^n(t_1, \ldots, t_n)$ is a term.
  (The superscript may be omitted if no confusion results.)
- 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.

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

Literals and Clauses

Literals: If P is an atomic formula,
then P and $\neg P$ are literals.

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.

4.3.2 Semantics of Clause-Form First-Order Predicate Logic

- Individual Constants, Function Symbols, Predicate Symbols, Ground Terms, and Ground Atomic Formulas as for Standard FOL.
- Skolem Constants are like indefinite individuals.
- Skolem Function Symbols are like indefinite function symbols.
- Ground Literals, Ground Clauses, and Sets of Clauses as for Clause-Form Propositional Logic.

Semantics of Open Clauses

If clause C contains variables $v_1, \ldots, v_n$,
then $C\{t_1/v_1, \ldots, t_n/v_n\}$ is a **ground instance** of C if it contains no more variables.

If C is an open clause,
$[\![C]\!]$ is True if every ground instance of C is True.
Otherwise, it is False.

That is, variables take on **universal interpretation**,
with scope being the clause.

4.3.3 Proof Theory of Clause-Form FOL

Notion of Proof: None!

Notion of Derivation: A set of clauses constitutes a derivation.

Assumptions: The derivation is initialized with a set of assumption clauses $A_1, \ldots, A_n$.

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

Derived Clause: If clause Q has been added to a set of clauses initialized with the set of assumption clauses $A_1, \ldots, A_n$ by one or more applications of resolution,
then $A_1, \ldots, A_n \vdash Q$.

Clause-Form FOL Rules of Inference
Version 1

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

Universal Instantion (temporary): $$\frac{C}{C\sigma}$$

Example Derivation

1.	$\{\neg Drives(x, y), Driver(x)\}$	$Assumption$
2.	$\{\neg Driver(z), \neg Passenger(z)\}$	$Assumption$
3.	$\{Drives(motherOf(Tom), Tom)\}$	$Assumption$
4.	$\{\neg Drives(motherOf(Tom), Tom), Driver(motherOf(Tom))\}$	$UI, 1, \{motherOf(Tom)/x, Tom/y\}$
5.	$\{Driver(motherOf(Tom))\}$	$R, 3, 4$
6.	$\{\neg Driver(motherOf(Tom)), \neg Passenger(motherOf(Tom))\}$	$UI, 2, \{motherOf(Tom)/z\}$
7.	$\{ \neg Passenger(motherOf(Tom))\}$	$R, 5, 6$

Motivation for a Shortcut

$$\begin{array}{cc}
\{P(x), L_1(x), \ldots, L_n(x)\} & \{\neg P(y), L_{n+1}(y), \ldots, L_m(y)\} \\[1em]
\downarrow \{a/x, a/y\} & \downarrow \{a/x, a/y\} \\[1em]
\{P(a), L_1(a), \ldots, L_n(a)\} & \{\neg P(a), L_{n+1}(a), \ldots, L_m(a)\} \\
\hline \\
\multicolumn{2}{c}{\{L_1(a), \ldots, L_n(a), L_{n+1}(a), \ldots, L_m(a)\}}
\end{array}$$

Most General Unifier

A most general unifier (**mgu**), of atomic formulas $\mathcal{A}$ and $\mathcal{B}$

is a substitution, μ,

such that $\mathcal{A}\mu = \mathcal{B}\mu =$ a common instance of $\mathcal{A}$ and $\mathcal{B}$

and such that every other common instance of $\mathcal{A}$ and $\mathcal{B}$ is an instance of it.

I.e., $\mathcal{A}\mu = \mathcal{B}\mu =$ a $\underline{\text{most general common instance}}$ of $\mathcal{A}$ and $\mathcal{B}$.

Example:
Unifier of $P(a, x, y)$ and $P(u, b, v)$ is $\{a/u, b/x, c/y, c/v\}$
giving $P(a, b, c)$
But more general is $\{a/u, b/x, y/v\}$ giving $P(a, b, y)$

Clause-Form FOL Rules of Inference
Version 2

Resolution:

$$\frac{\{A, L_1, \ldots, L_n\}, \{\neg B, L_{n+1}, \ldots, L_m\}}{\{L_1\mu, \ldots, L_n\mu, L_{n+1}\mu, \ldots, L_m\mu\}}$$

where μ is an mgu of A and B.

Assume two parent clauses have no variables in common.

Example Derivation Revisited

1. $\{\neg Drives(x, y), Driver(x)\}$ $\quad$ $Assumption$
2. $\{\neg Driver(z), \neg Passenger(z)\}$ $\quad$ $Assumption$
3. $\{Drives(motherOf(Tom), Tom)\}$ $\quad$ $Assumption$

4. $\{Driver(motherOf(Tom))\}$ $\quad$ $R, 1, 3, \{motherOf(Tom)/x, Tom/y\}$
5. $\{ \neg Passenger(motherOf(Tom))\}$ $\quad$ $R, 2, 4, \{motherOf(Tom)/z, \}$

Unification

To find the mgu of $\mathcal{A}$ and $\mathcal{B}$.

Some Examples:

$\mathcal{A}$	$\mathcal{B}$	mgu	mgci
$P(a,b)$	$P(a,b)$	$\{\}$	$P(a,b)$
$P(a)$	$P(b)$	*FAIL*	
$P(a,x)$	$P(y,b)$	$\{a/y, b/x\}$	$P(a,b)$
$P(a,x)$	$P(y,g(y))$	$\{a/y, g(a)/x\}$	$P(a,g(a))$
$P(x,f(x))$	$P(y,y)$	*FAIL* (*occurs check*)	

Substitution Composition

$P\sigma\tau = ((P\sigma)\tau) = P(\sigma \circ \tau)$

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

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

$\sigma \uplus \tau = \sigma \cup \{t/v \mid (t/v \in \tau) \wedge v \notin \sigma\}$

E.g.: $\{x/y, y/z\} \circ \{u/y, v/w\} = \{x/y, u/z, v/w\}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))      (P (f u) v v)`

μ `=` $\{\}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))`   `(P (f u) v v)`

μ = {}

...$\boxed{\texttt{x}}$ `(g x) (g (f a)))`   ...$\boxed{\texttt{(f u)}}$ `v v)`

$\mu = \{\}\circ\{\texttt{(f u)/x}\}=\{\texttt{(f u)/x}\}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))`   `(P (f u) v v)`

μ `=` $\{\}$

$\texttt{...}\boxed{\texttt{x}}\ \texttt{(g x) (g (f a)))}\qquad\texttt{...}\boxed{\texttt{(f u)}}\ \texttt{v v)}$

$\mu\ \texttt{=}\ \{\}\circ\{\texttt{(f u)/x}\}\texttt{=}\{\texttt{(f u)/x}\}$

$\texttt{...}\boxed{\texttt{(g x)}}\ \texttt{(g (f a)))}\qquad\texttt{...}\boxed{\texttt{v}}\ \texttt{v)}$

Manual Unification Algorithm

$\texttt{(P x (g x) (g (f a)))} \qquad \texttt{(P (f u) v v)}$

$\mu\ \texttt{=}\ \{\}$

$\texttt{...}\boxed{\texttt{x}}\ \texttt{(g x) (g (f a)))} \qquad \texttt{...}\boxed{\texttt{(f u)}}\ \texttt{v v)}$

$\mu\ \texttt{=}\ \{\}\circ\{\texttt{(f u)/x}\}\texttt{=}\{\texttt{(f u)/x}\}$

$\texttt{...}\boxed{\texttt{(g x)}}\ \texttt{(g (f a)))} \qquad \texttt{...}\boxed{\texttt{v}}\ \texttt{v)}$

$\texttt{...}\boxed{\texttt{(g (f u))}}\ \texttt{(g (f a)))} \qquad \texttt{...}\boxed{\texttt{v}}\ \texttt{v)}$

Manual Unification Algorithm

$\texttt{(P x (g x) (g (f a)))} \qquad \texttt{(P (f u) v v)}$

μ = {}

$\texttt{...}\boxed{\texttt{x}}\ \texttt{(g x) (g (f a)))} \qquad \texttt{...}\boxed{\texttt{(f u)}}\ \texttt{v v)}$

μ = {}∘{(f u)/x}={(f u)/x}

$\texttt{...}\boxed{\texttt{(g x)}}\ \texttt{(g (f a)))} \qquad \texttt{...}\boxed{\texttt{v}}\ \texttt{v)}$

$\texttt{...}\boxed{\texttt{(g (f u))}}\ \texttt{(g (f a)))} \qquad \texttt{...}\boxed{\texttt{v}}\ \texttt{v)}$

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

Manual Unification Algorithm

`(P x (g x) (g (f a)))`   `(P (f u) v v)`

μ = {}

$\texttt{...}\boxed{\texttt{x}}\texttt{ (g x) (g (f a)))}$ $\texttt{...}\boxed{\texttt{(f u)}}\texttt{ v v)}$

μ = {}∘{(f u)/x}={(f u)/x}

$\texttt{...}\boxed{\texttt{(g x)}}\texttt{ (g (f a)))}$ $\texttt{...}\boxed{\texttt{v}}\texttt{ v)}$

$\texttt{...}\boxed{\texttt{(g (f u))}}\texttt{ (g (f a)))}$ $\texttt{...}\boxed{\texttt{v}}\texttt{ v)}$

μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

$\texttt{...}\boxed{\texttt{(g (f a))}}\texttt{)}$ $\texttt{...}\boxed{\texttt{v}}\texttt{)}$

Manual Unification Algorithm

`(P x (g x) (g (f a)))` `(P (f u) v v)`

μ = {}

...$\boxed{\texttt{x}}$ `(g x) (g (f a)))` ...$\boxed{\texttt{(f u)}}$ `v v)`

μ = $\{\}\circ\{\texttt{(f u)/x}\}=\{\texttt{(f u)/x}\}$

...$\boxed{\texttt{(g x)}}$ `(g (f a)))` ...$\boxed{\texttt{v}}$ `v)`

...$\boxed{\texttt{(g (f u))}}$ `(g (f a)))` ...$\boxed{\texttt{v}}$ `v)`

μ = $\{\texttt{(f u)/x}\}\circ\{\texttt{(g (f u))/v}\} = \{\texttt{(f u)/x, (g (f u))/v}\}$

...$\boxed{\texttt{(g (f a))}}$`)` ...$\boxed{\texttt{v}}$`)`

...$\boxed{\texttt{(g (f a))}}$`)` ...$\boxed{\texttt{(g (f u))}}$`)`

Manual Unification Algorithm

`(P x (g x) (g (f a)))     (P (f u) v v)`

μ = {}

$\texttt{...}\boxed{\texttt{x}}\texttt{ (g x) (g (f a)))}\qquad\texttt{...}\boxed{\texttt{(f u)}}\texttt{ v v)}$

μ = {}$\circ${(f u)/x}={(f u)/x}

$\texttt{...}\boxed{\texttt{(g x)}}\texttt{ (g (f a)))}\qquad\texttt{...}\boxed{\texttt{v}}\texttt{ v)}$

$\texttt{...}\boxed{\texttt{(g (f u))}}\texttt{ (g (f a)))}\qquad\texttt{...}\boxed{\texttt{v}}\texttt{ v)}$

μ = {(f u)/x}$\circ${(g (f u))/v} = {(f u)/x, (g (f u))/v}

$\texttt{...}\boxed{\texttt{(g (f a))}}\texttt{)}\qquad\texttt{...}\boxed{\texttt{v}}\texttt{)}$

$\texttt{...}\boxed{\texttt{(g (f a))}}\texttt{)}\qquad\texttt{...}\boxed{\texttt{(g (f u))}}\texttt{)}$

$\texttt{...}\boxed{\texttt{a}}\texttt{)))}\qquad\texttt{...}\boxed{\texttt{u}}\texttt{)))}$

Manual Unification Algorithm

(P x (g x) (g (f a)))   (P (f u) v v)
μ = {}

...$\boxed{\texttt{x}}$ (g x) (g (f a)))   ...$\boxed{\texttt{(f u)}}$ v v)
μ = {}∘{(f u)/x}={(f u)/x}

...$\boxed{\texttt{(g x)}}$ (g (f a)))   ...$\boxed{\texttt{v}}$ v)
...$\boxed{\texttt{(g (f u))}}$ (g (f a)))   ...$\boxed{\texttt{v}}$ v)
μ = {(f u)/x}∘{(g (f u))/v} = {(f u)/x, (g (f u))/v}

...$\boxed{\texttt{(g (f a))}}$)   ...$\boxed{\texttt{v}}$)
...$\boxed{\texttt{(g (f a))}}$)   ...$\boxed{\texttt{(g (f u))}}$)
...$\boxed{\texttt{a}}$)))   ...$\boxed{\texttt{u}}$)))
μ = {(f u)/x, (g (f u))/v}∘{a/u} = {(f a)/x, (g (f a))/v, a/u}

(P x (g x) (g (f a)))μ = (P (f u) v v)μ = (P (f a) (g (f a)) (g (f a)))

Unification Algorithm

```
(defun unify (A B &optional mu)
  (cond ((eql mu 'FAIL) 'FAIL)
        ((eql A B) mu)
        ((variablep A) (unifyVar A B mu))
        ((variablep B) (unifyVar B A mu))
        ((or (atom A) (atom B)) 'FAIL)
        ((/= (length A) (length B)) 'FAIL)
        (t (unify (rest A)
                  (rest B)
                  (unify (first A) (first B) mu)))))
```

Note: a more efficient version is implemented in `prover.cl`

UnifyVar

```
(defun unifyVar (var term subst)
  (if (var-in-substp var subst)
      (unify (term-of-var-in-subst var subst) term subst)
    (let ((newterm (apply-sub subst term)))
      (cond ((eql var newterm) subst)
            ((occursIn var newterm) 'FAIL)
            (t (compose subst
                        (list (pair newterm var))))))))
```

Program Assertion

If original $\mathcal{A}$ and $\mathcal{B}$ have no variables in common,

then throughout the above program

no substitution will have one of its variables occurring in one of its terms.

Therefore, for any expression $\mathcal{E}$ and any substitution σ formed in the above program, $\mathcal{E}\sigma\sigma = \mathcal{E}\sigma$.

4.3.4 Resolution Refutation Example

To decide if

$\{\neg Drives(x, y), Driver(x)\}, \{\neg Driver(x), \neg Passenger(x)\},$
$\{Drives(motherOf(Tom), Betty)\}$
$\models \{\neg Passenger(motherOf(Tom))\}$

1.	$\{\neg Drives(x_1, y_1), Driver(x_1)\}$	Assumption
2.	$\{\neg Driver(x_2), \neg Passenger(x_2)\}$	Assumption
3.	$\{Drives(motherOf(Tom), Betty)\}$	Assumption
5.	$\{Passenger(motherOf(Tom))\}$	From query
6.	$\{\neg Driver(motherOf(Tom))\}$	$R, 2, 5, \{motherOf(Tom)/x_2\}$
7.	$\{\neg Drives(motherOf(Tom), y_7)\}$	$R, 1, 6, \{motherOf(Tom)/x_1\}$
8.	$\{\}$	$R, 3, 7, \{Betty/y_7\}$

Example Using prover

```
prover(21): (prove '((or (not (Drives ?x ?y)) (Driver ?x))
                     (or (not (Driver ?x)) (not (Passenger ?x)))
                     (Drives (motherOf Tom) Betty))
                   '(not (Passenger (motherOf Tom))))

 1  ((Drives (motherOf Tom) Betty)) Assumption
 2  ((not (Drives ?3 ?5)) (Driver ?3))  Assumption
 3  ((not (Driver ?9)) (not (Passenger ?9)))  Assumption
 4  ((Passenger (motherOf Tom)))  From Query
 5  ((not (Driver (motherOf Tom)))) R,4,3,{(motherOf Tom)/?9}
 6  ((not (Drives (motherOf Tom) ?86))) R,5,2,{(motherOf Tom)/?3}
 7  nil             R,6,1,{Betty/?86}
QED
```

Example Using `snark`

```
snark-user(84): (initialize)
snark-user(85): (assert '(or (not (Drives ?x ?y)) (Driver ?x)))
snark-user(86): (assert '(or (not (Driver ?x))
                             (not (Passenger ?x))))
snark-user(87): (assert '(Drives (motherOf Tom) Betty))
snark-user(88): (prove '(not (Passenger (motherOf Tom))))
(Refutation
(Row 1 (or (not (Drives ?x ?y)) (Driver ?x)) assertion)
(Row 2 (or (not (Driver ?x)) (not (Passenger ?x))) assertion)
(Row 3 (Drives (motherOf Tom) Betty) assertion)
(Row 4 (Passenger (motherOf Tom)) ~conclusion)
(Row 5 (not (Driver (motherOf Tom))) (resolve 2 4))
(Row 6 (not (Drives (motherOf Tom) ?x)) (resolve 5 1))
(Row 7 false (resolve 6 3))
)
:proof-found
```

Resolution Refutation is Incomplete for FOL

1. $\{P(u), P(v)\}$
2. $\{\neg P(x), \neg P(y)\}$
3. $\{P(w), \neg P(z)\}$ $\quad R, 1, 2, \{u/x, w/v, z/y\}$

$\vdots$

Clause-Form FOL Rules of Inference
Version 3 (Last)

Resolution: $$\frac{\{A, L_1, \ldots, L_n\}, \{\neg B, L_{n+1}, \ldots, L_m\}}{\{L_1\mu, \ldots, L_n\mu, L_{n+1}\mu, \ldots, L_m\mu\}}$$

where μ is an mgu of A and B.

Factoring: $$\frac{\{A, B, L_1, \ldots, L_n\}}{\{A\mu, L_1\mu, \ldots, L_n\mu\}}$$

where μ is an mgu of A and B.
(Note: Special case of UI.)

Resolution Refutation with Factoring is Complete

If $A_1, \ldots, A_n \models Q$, then $A_1, \ldots, A_n, \neg Q \vdash_{R+F} \{\}$.

For example,

1.	$\{P(u), P(v)\}$	
2.	$\{\neg P(x), \neg P(y)\}$	
3.	$\{P(w)\}$	$F, 1, \{w/u, w/v\}$
4.	$\{\neg P(z)\}$	$F, 2, \{z/x, z/y\}$
5.	$\{\}$	$R, 3, 4, \{w/z\}$

However, resolution refutation with factoring is still not a decision procedure—it is a semi-decision procedure.

Factoring (Condensing) by `snark`

```
snark-user(30): (initialize)
; Running SNARK from ...
nil
snark-user(31): (assert '(or (P ?u) (P ?v)))
nil
snark-user(32): (prove '(and (P ?x) (P ?y)))

(Refutation
(Row 1
   (or (P ?x) (P ?y))
   assertion)
(Row 2
   (P ?x)
   (condense 1))
(Row 3
   (or (not (P ?x)) (not (P ?y)))
   negated_conjecture)
(Row 4
   false
   (rewrite 3 2))
)

:proof-found
```

SNARK has both factoring and condensing, which is factoring combined with immediate subsumption elimination when the factored clause subsumes the original clause. The clause '(or (p a ?x) (p ?y b)) gets factored, but not condensed. [Mark Stickel, personal communication, March, 2008]

Efficiency Rules

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

Pure-Literal Elimination: If clause C contains a literal A ($\neg A$) and no clause contains a literal $\neg B$ (B) such that A and B are unifiable, delete C from the set of clauses. (Check throughout.)

Subsumption Elimination: If the set of clauses contains clauses C_1 and C_2 such that there is a substitution σ for which $C_1\sigma \subseteq C_2$, delete C_2 from the set of clauses. (Check throughout.)

These rules delete unhelpful clauses.

Subsumption may be required to cut infinite loops.

Subsumption Cutting a Loop

```
prover(22): (prove '((if (and (ancestor ?x ?y)
                              (ancestor ?y ?z))
                         (ancestor ?x ?z)))
                   '(ancestor ?x stu))

 1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
     (ancestor ?0 ?2))  Assumption
 2  ((not (ancestor ?3 stu))) From Query
```

Initial Resolution Steps

```
1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
    (ancestor ?0 ?2))                                         Assumption
2  ((not (ancestor ?3 stu)))                                  From Query
3  ((not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
                                                     R,2,1,{stu/?2, ?0/?3}
4  ((not (ancestor ?6 stu)) (not (ancestor ?7 ?8))
    (not (ancestor ?8 ?6)))                          R,3,1,{?2/?5, ?0/?4}
5  ((not (ancestor ?9 ?10)) (not (ancestor ?10 ?11))
    (not (ancestor ?11 stu)))                          R,3,1,{stu/?2, ?0/?5}
              .
              .
              .
```

Subsumption Cuts the Loop

```
 1  ((not (ancestor ?0 ?1)) (not (ancestor ?1 ?2))
     (ancestor ?0 ?2))                                  Assumption
 2  ((not (ancestor ?3 stu)))                           From Query
 3  (not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
                                                   R,2,1,{stu/?2, ?0/?3}
 4  ((not (ancestor stu stu)))  F,3,{stu/?5, stu/?4}
Deleting  4 ((not (ancestor stu stu)))
 because it's subsumed by  2 ((not (ancestor ?3 stu)))
Deleting  3 ((not (ancestor ?4 ?5)) (not (ancestor ?5 stu)))
 because it's subsumed by  2 ((not (ancestor ?3 stu)))
nil
```

Strategies

Unit Preference: Resolve shorter clauses before longer clauses.
Least Symbol Count Version: Count symbols, not literals.

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

Many others

These are heuristics for finding {} faster.

Least Symbol Count Version of Unit Preference

Instead of counting literals,

count symbols

ignoring negation operator.

Equivalent to standard unit preference for Propositional Logic.

Problem with
Literal-Counting Unit Preference

```
1(1/2)  ((walkslikeduck daffy)) Assumption
2(1/2)  ((talkslikeduck daffy)) Assumption
3(2/5)  ((not (duck (motherof ?1))) (duck ?1))  Assumption
4(3/6)  ((not (walkslikeduck ?3)) (not (talkslikeduck ?3)) (duck ?3)) Assumption
5(1/2)  ((not (duck daffy)))  From Query
6(1/3)  ((not (duck (motherof daffy)))) R,5,3,{daffy/?1}
7(1/4)  ((not (duck (motherof (motherof daffy)))))  R,6,3,{(motherof daffy)/?1}
8(1/5)  ((not (duck
               (motherof
                (motherof
                 (motherof daffy)))))) R,7,3,{(motherof (motherof daffy))/?1}
9(1/6)  ((not (duck
               (motherof
                (motherof
                 (motherof
                  (motherof
                   daffy)))))))  R,8,3,{(motherof (motherof (motherof daffy)))/?1}
```

Solution with
Least Symbol Count Version

```
 1(1/2)  ((walkslikeduck daffy)) Assumption
 2(1/2)  ((talkslikeduck daffy)) Assumption
 3(2/5)  ((not (duck (motherof ?5))) (duck ?5))  Assumption
 4(3/6)  ((not (walkslikeduck ?13)) (not (talkslikeduck ?13)) (duck ?13)) Assumption
 5(1/2)  ((not (duck daffy)))  From Query

 6(1/3)  ((not (duck (motherof daffy)))) R,5,3,{daffy/?1}
 7(1/4)  ((not (duck (motherof (motherof daffy)))))  R,6,3,{(motherof daffy)/?1}
 8(2/4)  ((not (walkslikeduck daffy)) (not (talkslikeduck daffy))) R,5,4,{daffy/?3}
 9(1/2)  ((not (talkslikeduck daffy))) R,8,1,{}
10(0/0)  nil             R,9,2,{}
QED
```

4.4 Translating Standard FOL Wffs into FOL Clause Form

Useful Meta-Theorems

- If A is (an occurrence of) a subformula of B,
  and $\models A \Leftrightarrow C$,
  then $\models B \Leftrightarrow B\{C/A\}$
- $\forall x_1(\cdots \forall x_n(\cdots \exists y A(x_1, \ldots, x_n, y) \cdots) \cdots)$ is consistent
  if and only if
  $\forall x_1(\cdots \forall x_n(\cdots A(x_1, \ldots, x_n, f^n(x_1, \ldots, x_n)) \cdots) \cdots)$
  is consistent,
  where f^n is a new Skolem function.
  Note: use a new Skolem constant instead of $f^0()$.

Translating Standard FOL Wffs into FOL Clause Form
Step 1

Eliminate occurrences of $\Leftrightarrow$ using

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

From:
$\forall x[Parent(x) \Leftrightarrow (Person(x) \land \exists y(Person(y) \land childOf(y,x)))]$

To:
$\forall x[(Parent(x) \Rightarrow (Person(x) \land \exists y(Person(y) \land childOf(y,x))))$
$\quad \land((Person(x) \land \exists y(Person(y) \land childOf(y,x))) \Rightarrow Parent(x))]$

Translation Step 2

Eliminate occurrences of ⇒ using

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

From:

$$\forall x[(Parent(x) \Rightarrow (Person(x) \land \exists y(Person(y) \land childOf(y,x))))$$
$$\land((Person(x) \land \exists y(Person(y) \land childOf(y,x))) \Rightarrow Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \lor (Person(x) \land \exists y(Person(y) \land childOf(y,x))))$$
$$\land(\neg(Person(x) \land \exists y(Person(y) \land childOf(y,x))) \lor Parent(x))]$$

Translation Step 3

Translate to *miniscope* form using

$$\models \neg\neg A \Leftrightarrow A$$

$$\models \neg(A \land B) \Leftrightarrow (\neg A \lor \neg B) \qquad \models \neg(A \lor B) \Leftrightarrow (\neg A \land \neg B)$$

$$\models \neg\forall x A(x) \Leftrightarrow \exists x \neg A(x) \qquad \models \neg\exists x A(x) \Leftrightarrow \forall x \neg A(x)$$

From:

$$\forall x[(\neg Parent(x) \lor (Person(x) \land \exists y(Person(y) \land childOf(y,x))))$$
$$\quad \land(\neg(Person(x) \land \exists y(Person(y) \land childOf(y,x))) \lor Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \lor (Person(x) \land \exists y(Person(y) \land childOf(y,x))))$$
$$\quad \land((\neg Person(x) \lor \forall y(\neg Person(y) \lor \neg childOf(y,x))) \lor Parent(x))]$$

Translation Step 4

Rename apart: If any two quantifiers bind the same variable, rename all occurrences of one of them.

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x)))) \\ \wedge((\neg Person(x) \vee \forall y(\neg Person(y) \vee \neg childOf(y,x))) \vee Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x)))) \\ \wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

Optional Translation Step 4.5

Translate into Prenex Normal Form using:

$$\models (A \wedge \forall x B(x)) \Leftrightarrow \forall x(A \wedge B(x)) \quad \models (A \wedge \exists x B(x)) \Leftrightarrow \exists x(A \wedge B(x))$$

$$\models (A \vee \forall x B(x)) \Leftrightarrow \forall x(A \vee B(x)) \quad \models (A \vee \exists x B(x)) \Leftrightarrow \exists x(A \vee B(x))$$

as long as x does not occur free in A.

From:

$$\begin{aligned}&\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))\\ &\quad \wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]\end{aligned}$$

To:

$$\begin{aligned}&\forall x\exists y\forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(y) \wedge childOf(y,x))))\\ &\qquad\quad \wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]\end{aligned}$$

Translation Step 5

Skolemize

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge \exists y(Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

To:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x),x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

or

From:

$$\forall x \exists y \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(y) \wedge childOf(y,x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

To:

$$\forall x \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x),x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z,x))) \vee Parent(x))]$$

Translation Step 6

Discard all occurrences of "$\forall x$" for any variable x.

From:

$$\forall x[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee \forall z(\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

Or from:

$$\forall x \forall z[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

To:

$$[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$$
$$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$$

Translation Step 7

CNF: Translate into Conjunctive Normal Form, using

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

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

From:

$[(\neg Parent(x) \vee (Person(x) \wedge (Person(f(x)) \wedge childOf(f(x), x))))$

$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$

To:

$[((\neg Parent(x) \vee Person(x))$

$\wedge((\neg Parent(x) \vee Person(f(x)))$

$\wedge(\neg Parent(x) \vee childOf(f(x), x)))))$

$\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]$

Translation Step 8

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

From:

$$
\begin{aligned}
&[((\neg Parent(x) \vee Person(x))\\
&\quad \wedge((\neg Parent(x) \vee Person(f(x)))\\
&\qquad \wedge(\neg Parent(x) \vee childOf(f(x), x)))))\\
&\;\wedge((\neg Person(x) \vee (\neg Person(z) \vee \neg childOf(z, x))) \vee Parent(x))]
\end{aligned}
$$

To:

$$
\begin{aligned}
&[(\neg Parent(x) \vee Person(x))\\
&\;\wedge(\neg Parent(x) \vee Person(f(x)))\\
&\;\wedge(\neg Parent(x) \vee childOf(f(x), x))\\
&\;\wedge(\neg Person(x) \vee \neg Person(z) \vee \neg childOf(z, x) \vee Parent(x))]
\end{aligned}
$$

Translation Step 9

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

From:

$$[(\neg Parent(x) \vee Person(x))$$
$$\wedge(\neg Parent(x) \vee Person(f(x)))$$
$$\wedge(\neg Parent(x) \vee childOf(f(x), x))$$
$$\wedge(\neg Person(x) \vee \neg Person(z) \vee \neg childOf(z, x) \vee Parent(x))]$$

To:

$$\{\{\neg Parent(x), Person(x)\},$$
$$\{\neg Parent(x), Person(f(x))\},$$
$$\{\neg Parent(x), childOf(f(x), x)\},$$
$$\{\neg Person(x), \neg Person(z), \neg childOf(z, x), Parent(x)\}\}$$

Translation Step 10

Rename the clauses apart
so that no variable occurs in more than one clause.

From:

$$\{\{\neg Parent(x), Person(x)\},$$
$$\{\neg Parent(x), Person(f(x))\},$$
$$\{\neg Parent(x), childOf(f(x), x)\},$$
$$\{\neg Person(x), \neg Person(z), \neg childOf(z, x), Parent(x)\}\}$$

To:

$$\{\{\neg Parent(x_1), Person(x_1)\},$$
$$\{\neg Parent(x_2), Person(f(x_2))\},$$
$$\{\neg Parent(x_3), childOf(f(x_3), x_3)\},$$
$$\{\neg Person(x_4), \neg Person(z_4), \neg childOf(z_4, x_4), Parent(x_4)\}\}$$

Use of Translation

$$A_1, \ldots, A_n \models B$$

iff

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

Example with `ubprover`

```
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(Parent Betty))

 1  ((Person Tom))                               Assumption
 2  ((Person Betty))                             Assumption
 3  ((childOf Tom Betty))                        Assumption
 4  ((not (Parent ?4)) (Person ?4))              Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))         Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))     Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))          Assumption
 8  ((not (Parent Betty)))                       From Query
```

Resolution Steps

```
 1  ((Person Tom))                         Assumption
 2  ((Person Betty))                       Assumption
 3  ((childOf Tom Betty))                  Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))    Assumption
 8  ((not (Parent Betty)))                 From Query
 9  ((not (Person Betty)) (not (Person ?9))
     (not (childOf ?9 Betty)))             R,8,7,{Betty/?7}
13  ((not (Person Betty))
     (not (childOf Tom Betty)))            R,9,1,{Tom/?9}
14  ((not (childOf Tom Betty)))            R,13,2,{}
15  nil                                    R,14,3,{}
QED
```

Example with `SNARK`

```
snark-user(42): (initialize)
; Running SNARK from ...
nil
snark-user(43): (assert
                 '(forall (x)
                   (iff (Parent x)
                        (and (Person x)
                             (exists (y)
                              (and (Person y) (childOf y x)))))))
nil
snark-user(44): (assert '(Person Tom))
nil
snark-user(45): (assert '(Person Betty))
nil
snark-user(46): (assert '(childOf Tom Betty))
nil
snark-user(47): (prove '(Parent Betty))
```

Initial Set of Clauses

```
(Row 1 (or (not (Parent ?x)) (Person ?x)) assertion)
(Row 2 (or (not (Parent ?x)) (Person (skolembiry1 ?x))) assertion)
(Row 3 (or (not (Parent ?x)) (childOf (skolembiry1 ?x) ?x)) assertion)
(Row 4 (or (Parent ?x) (not (Person ?x)) (not (Person ?y)) (not (childOf ?y ?
       assertion)
(Row 5 (Person Tom) assertion)
(Row 6 (Person Betty) assertion)
(Row 7 (childOf Tom Betty) assertion)
(Row 8 (not (Parent Betty)) negated_conjecture)
(Row 9 (or (not (Person ?x)) (not (childOf ?x Betty))) (rewrite (resolve 8 4)
(Row 10 false (rewrite (resolve 9 7) 5))
```

Refutation

```
(Refutation
(Row 4 (or (Parent ?x) (not (Person ?x)) (not (Person ?y)) (not (childOf ?y ?
       assertion)
(Row 5 (Person Tom) assertion)
(Row 6 (Person Betty) assertion)
(Row 7 (childOf Tom Betty) assertion)
(Row 8 (not (Parent Betty)) negated_conjecture)
(Row 9 (or (not (Person ?x)) (not (childOf ?x Betty))) (rewrite (resolve 8 4)
(Row 10 false (rewrite (resolve 9 7) 5))
)
:proof-found
```

A `ubprover` Example Using the Skolem Function

```
prover(72): (prove
            '((forall x (iff (Parent x)
                             (and (Person x)
                                  (exists y (and (Person y) (childOf y x))))))
              (Person Tom) (Person Betty) (Parent Betty))
            '(exists x (childOf x Betty)))

 1  ((Person Tom))                            Assumption
 2  ((Person Betty))                          Assumption
 3  ((Parent Betty))                          Assumption
 4  ((not (Parent ?4)) (Person ?4))           Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))      Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))  Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))       Assumption
 8  ((not (childOf ?10 Betty)))               From Query
 9  ((not (Parent Betty)))                    R,8,6,{Betty/?6, (S3 Betty)/?10}
10  nil                                       R,9,3,{}
QED
```

4.5 Asking Wh Questions

Given

$$\forall x[Parent(x) \Leftrightarrow (Person(x) \land \exists y(Person(y) \land childOf(y, x)))]$$

$$Person(Tom)$$

$$Person(Betty)$$

$$childOf(Tom, Betty)$$

Ask: “Who is a parent?”

Answer via constructive proof of $\exists x \; Parent(x)$.

Try to Answer Wh Question

```
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(exists x (Parent x)))


 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((Parent Betty))                            Assumption
 4  ((not (Parent ?4)) (Person ?4))             Assumption
 5  ((not (Parent ?5)) (Person (S3 ?5)))        Assumption
 6  ((not (Parent ?6)) (childOf (S3 ?6) ?6))  Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))         Assumption
 8  ((not (childOf ?10 Betty)))                 From Query
```

Resolution Steps

```
 1  ((Person Tom))                                Assumption
 2  ((Person Betty))                              Assumption
 3  ((childOf Tom Betty))                         Assumption
 7  ((not (Person ?7)) (not (Person ?8))
     (not (childOf ?8 ?7)) (Parent ?7))           Assumption
 8  ((not (Parent ?10)))                          From Query
 9  ((not (Person ?11)) (not (Person ?12))
     (not (childOf ?12 ?11)))                     R,8,7,{?7/?10}
15  ((not (Person ?16)) (not (childOf Tom ?16)))  R,9,1,{Tom/?12}
16  ((not (childOf Tom Tom)))                     R,15,1,{Tom/?16}
17  ((not (childOf Tom Betty)))                   R,15,2,{Betty/?16}
18  nil                                           R,17,3,{}
QED
```

The Answer Predicate

Instead of query $\exists x_1 \cdots \exists x_n P(x_1, \ldots, x_n)$,

and resolution refutation with $\{\neg P(x_1, \ldots, x_n)\}$

until $\{\}$,

use $\forall x_1 \cdots \forall x_n (P(x_1, \ldots, x_n) \Rightarrow \mathit{Answer}(P(x_1, \ldots, x_n)))$

and do direct resolution with

$$\{\neg P(x_1, \ldots, x_n), \mathit{Answer}(P(x_1, \ldots, x_n))\}$$

until $\{(\mathit{Answer} \ldots) \cdots (\mathit{Answer} \ldots)\}$.

General Procedure for Inserting The Answer Predicate

Let:

Q be either $\forall$ or $\exists$;

$\overline{Q}$ be either $\exists$ or $\forall$, respectively;

Prenex Normal form of query be $Q_1 x_1 \cdots Q_n x_n P(x_1, \ldots, x_n)$.

Do direct resolution with clause form of
$\overline{Q_1} x_1 \cdots \overline{Q_n} x_n (P(x_1, \ldots, x_n) \Rightarrow Answer(P(x_1, \ldots, x_n)))$
until generate $\{(Answer \ldots) \cdots (Answer \ldots)\}$.

Using the Answer Predicate

```
(setf *UseAnswer* t)
(prove
 '((forall x (iff (Parent x)
                  (and (Person x)
                       (exists y (and (Person y) (childOf y x))))))
   (Person Tom) (Person Betty) (childOf Tom Betty))
 '(exists x (Parent x)))

 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((childOf Tom Betty))                       Assumption
 4  ((not (Parent ?3)) (Person ?3))             Assumption
 5  ((not (Parent ?4)) (Person (S2 ?4)))        Assumption
 6  ((not (Parent ?5)) (childOf (S2 ?5) ?5))  Assumption
 7  ((not (Person ?6)) (not (Person ?7))
     (not (childOf ?7 ?6)) (Parent ?6))        Assumption
 8  ((not (Parent ?9)) (Answer (Parent ?9)))  From Query
```

Resolution Steps

```
 1  ((Person Tom))                              Assumption
 2  ((Person Betty))                            Assumption
 3  ((childOf Tom Betty))                       Assumption
 7  ((not (Person ?6)) (not (Person ?7))
     (not (childOf ?7 ?6)) (Parent ?6))         Assumption
 8  ((not (Parent ?9)) (Answer (Parent ?9)))    From Query
 9  ((Answer (Parent ?10)) (not (Person ?10))
     (not (Person ?11)) (not (childOf ?11 ?10))) R,8,7,{?6/?9}
15  ((Answer (Parent Betty))
     (not (Person Betty)) (not (Person Tom)))   R,9,3,{Betty/?10,
                                                       Tom/?11}
26  ((Answer (Parent Betty)) (not (Person Tom))) R,15,2,{}
29  ((Answer (Parent Betty)))                   R,26,1,{}
QED
```

Answer Predicate in snark

```
snark-user(11): (assert '(forall x (iff (Parent x)
                                   (exists y (and (Person y)
                                                  (childOf y x))))))
nil
snark-user(12): (assert '(Person Tom))
nil
snark-user(13): (assert '(Person Betty))
nil
snark-user(14): (assert '(childOf Tom Betty))
nil
snark-user(15): (prove '(exists x (Parent x))
                       :answer '(Parent x))
```

snark Refutation

```
(Refutation
(Row 3
   (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (childOf Tom Betty) assertion)
(Row 7 (not (Parent ?x)) negated_conjecture
   Answer (Parent ?x))
(Row 8 (or (not (Person ?x)) (not (childOf ?x ?y))) (resolve 7 3)
   Answer (Parent ?y))
(Row 9 false (rewrite (resolve 8 6) 4)
   Answer (Parent Betty))
)
:proof-found
```

Answer Predicate with `ask`

From same SNARK KB:

```
snark-user(18): (ask '(exists x (Parent x)) :answer '(Parent x))
(Parent Betty)
```

Using :printProof

```
snark-user(19): (ask '(Parent ?x) :answer '(Parent ?x)
                                  :printProof t)
(Refutation
(Row 3 (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4 (Person Tom) assertion)
(Row 6 (childOf Tom Betty) assertion)
(Row 13 (not (Parent ?x)) negated_conjecture
   Answer (Parent ?x))
(Row 14 (or (not (Person ?x)) (not (childOf ?x ?y)))
   (resolve 13 3)
   Answer (Parent ?y))
(Row 15 false (rewrite (resolve 14 6) 4)
   Answer (Parent Betty))
)
(Parent Betty)
```

Answer Predicate with `query`

From same SNARK KB:

```
snark-user(9): (query "Who is a parent?"
                      '(exists x (Parent x))
                      :answer '(Parent x))

Who is a parent?
  (ask '(exists x (Parent x))) = (Parent Betty)
```

query **with** :answer **and** :printProof

```
snark-user(10): (query "Who is a parent?"
'(exists x (Parent x)) :answer '(Parent x) :printProof t)
Who is a parent?
(Refutation
(Row 3
   (or (Parent ?x) (not (Person ?y)) (not (childOf ?y ?x)))
   assertion)
(Row 4
   (Person Tom)
   assertion)
(Row 6
   (childOf Tom Betty)
   assertion)
(Row 19
   (not (Parent ?x))
   negated_conjecture
   Answer (Parent ?x))
(Row 20
   (or (not (Person ?x)) (not (childOf ?x ?y)))
   (resolve 19 3)
   Answer (Parent ?y))
(Row 21
   false
   (rewrite (resolve 20 6) 4)
   Answer (Parent Betty))
)
  (ask '(exists x (Parent x))) = (Parent Betty)
```

Disjunctive Answers

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         (or (Red b) (Green b)))
       '(exists (x y)
               (and (Red x) (Green y) (On x y))))
 1  ((On a b))                                         Assumption
 2  ((On b c))                                         Assumption
 3  ((Red a))                                          Assumption
 4  ((Green c))                                        Assumption
 5  ((Red b) (Green b))                                Assumption
 6  ((not (Red ?28)) (not (Green ?30))
     (not (On ?28 ?30))
     (Answer (and (Red ?28) (Green ?30) (On ?28 ?30)))) From Query
```

Resolution Steps

```
 9  ((Answer (and (Red a) (Green ?107) (On a ?107)))
     (not (On a ?107)) (not (Green ?107)))            R,6,3,{a/?28}

10  ((Answer (and (Red ?112) (Green c) (On ?112 c)))
     (not (On ?112 c)) (not (Red ?112)))              R,6,4,{c/?30}

11  ((Answer (and (Red b) (Green ?117) (On b ?117)))
     (not (On b ?117)) (not (Green ?117)) (Green b))  R,6,5,{b/?28}

13  ((not (Red b))
     (Answer (and (Red b) (Green c) (On b c))))       R,10,2,{b/?112}
```

Resolution Finished

```
16  ((Answer (and (Red b) (Green c) (On b c)))
     (Green b))                                     R,13,5,{}

20  ((not (On a b))
     (Answer (and (Red a) (Green b) (On a b)))
     (Answer (and (Red b) (Green c) (On b c)))) R,9,16,{b/?107}

22  ((Answer (and (Red b) (Green c) (On b c)))
     (Answer (and (Red a) (Green b) (On a b)))) R,20,1,{}
QED
```

Multiple Clauses From Query

```
(prove '((On a b)(On b c)
         (Red a) (Green c)
         (or (Red b) (Green b)))
       '(exists x (or (Red x) (Green x))))

 1  ((On a b))                           Assumption
 2  ((On b c))                           Assumption
 3  ((Red a))                            Assumption
 4  ((Green c))                          Assumption
 5  ((Red b) (Green b))                  Assumption
 6  ((not (Red ?25))
     (Answer (or (Red ?25) (Green ?25)))) From Query
 7  ((not (Green ?27))
     (Answer (or (Red ?27) (Green ?27)))) From Query
 8  ((Answer (or (Red a) (Green a))))     R,6,3,{a/?25}
QED
```

Resolution Produces Only 1 Answer

```
snark-user(20): (initialize)
; Running SNARK from ...
nil

snark-user(21): (assert ’(Man Socrates))
nil

snark-user(22): (assert ’(Man Turing))
nil

snark-user(23): (ask ’(Man ?x) :answer ’(One man is ?x))
(One man is Turing)
```

Generic and Hypothetical Answers

Every clause that descends from a query clause (that contains an `Answer` predicate) is an answer of some sort.[a]

---

[a] Debra T. Burhans and Stuart C. Shapiro, Defining Answer Classes Using Resolution Refutation, *Journal of Applied Logic 5*, 1 (March 2007), 70–91 . `http://www.cse.buffalo.edu/~shapiro/Papers/bursha05.pdf`

Example of Generic and Hypothetical Answers Question

```
(prove '((forall (x y z) (if (and (Member x FBS) (Sport y)
                                  (Athlete z) (PlaysWell z y))
                             (ProvidesScholarshipFor x z)))
         (forall x (if (Sport x) (Activity x)))
         (forall x (if (Activity x) (or (Sport x) (Game x))))
         (forall x (if (or (Member x MAC) (Member x Big10) (Member Pac10 x))
                       (Member x FBS)))
         (Member Buffalo MAC) (Member KentSt MAC)
         (Member Wisconsin Big10) (Member Indiana Big10)
         (Member Stanford Pac10) (Member Berkeley Pac10)
         (Activity Frisbee))

       '(exists x (ProvidesScholarshipFor Buffalo x)))
```

Initial Clauses

```
 1  ((Member Buffalo MAC))                                     Assumption
 2  ((Member KentSt MAC))                                      Assumption
 3  ((Member Wisconsin Big10))                                 Assumption
 4  ((Member Indiana Big10))                                   Assumption
 5  ((Member Stanford Pac10))                                  Assumption
 6  ((Member Berkeley Pac10))                                  Assumption
 7  ((Activity Frisbee))                                       Assumption
 8  ((not (Sport ?7)) (Activity ?7))                           Assumption
 9  ((not (Member ?11 MAC)) (Member ?11 FBS))                  Assumption
10  ((not (Member ?12 Big10)) (Member ?12 FBS))                Assumption
11  ((not (Member Pac10 ?13)) (Member ?13 FBS))                Assumption
12  ((not (Activity ?9)) (Sport ?9) (Game ?9))                 Assumption
13  ((not (Member ?3 FBS)) (not (Sport ?4)) (not (Athlete ?5))
     (not (PlaysWell ?5 ?4)) (ProvidesScholarshipFor ?3 ?5))  Assumption
14  ((not (ProvidesScholarshipFor Buffalo ?15))
     (Answer (ProvidesScholarshipFor Buffalo ?15)))           From Query
```

Resolvents

```
15  ((Answer (ProvidesScholarshipFor Buffalo ?16)) (not (Member Buffalo FBS)) (not (Sport ?17))
     (not (Athlete ?16)) (not (PlaysWell ?16 ?17)))  R,14,13,{?5/?15, Buffalo/?3}
16  ((not (Member Buffalo MAC)) (Answer (ProvidesScholarshipFor Buffalo ?18)) (not (Sport ?19))
     (not (Athlete ?18)) (not (PlaysWell ?18 ?19)))  R,15,9,{Buffalo/?11}
17  ((not (Member Buffalo Big10)) (Answer (ProvidesScholarshipFor Buffalo ?20)) (not (Sport ?21))
     (not (Athlete ?20)) (not (PlaysWell ?20 ?21)))  R,15,10,{Buffalo/?12}
18  ((not (Member Pac10 Buffalo)) (Answer (ProvidesScholarshipFor Buffalo ?22)) (not (Sport ?23))
     (not (Athlete ?22)) (not (PlaysWell ?22 ?23)))  R,15,11,{Buffalo/?13}
19  ((Game ?24) (not (Activity ?24)) (Answer (ProvidesScholarshipFor Buffalo ?25))
     (not (Member Buffalo FBS)) (not (Athlete ?25)) (not (PlaysWell ?25 ?24)))  R,15,12,{?9/?17}
20  ((Game ?26) (not (Activity ?26)) (not (Member Pac10 Buffalo)) (Answer (ProvidesScholarshipFor Buffalo ?27))
     (not (Athlete ?27)) (not (PlaysWell ?27 ?26)))  R,18,12,{?9/?23}
21  ((Game ?28) (not (Activity ?28)) (not (Member Buffalo Big10)) (Answer (ProvidesScholarshipFor Buffalo ?29))
     (not (Athlete ?29)) (not (PlaysWell ?29 ?28)))  R,17,12,{?9/?21}
22  ((Answer (ProvidesScholarshipFor Buffalo ?30)) (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31))) R,16,1,{}
23  ((Game ?32) (not (Activity ?32)) (not (Member Buffalo MAC)) (Answer (ProvidesScholarshipFor Buffalo ?33))
     (not (Athlete ?33)) (not (PlaysWell ?33 ?32)))  R,16,12,{?9/?19}
24  ((Game ?34) (not (Activity ?34)) (Answer (ProvidesScholarshipFor Buffalo ?35)) (not (Athlete ?35))
     (not (PlaysWell ?35 ?34))) R,22,12,{?9/?31}
25  ((Game Frisbee) (Answer (ProvidesScholarshipFor Buffalo ?36)) (not (Athlete ?36))
     (not (PlaysWell ?36 Frisbee)))  R,24,7,{Frisbee/?34}
26  ((not (Sport ?37)) (Game ?37) (Answer (ProvidesScholarshipFor Buffalo ?38)) (not (Athlete ?38))
     (not (PlaysWell ?38 ?37)))  R,24,8,{?7/?34}
nil
```

Non-Subsumed Resolvents

```
22  ((Answer (ProvidesScholarshipFor Buffalo ?30))
     (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31)))

24  ((Game ?34) (not (Activity ?34))
     (Answer (ProvidesScholarshipFor Buffalo ?35))
     (not (Athlete ?35)) (not (PlaysWell ?35 ?34)))

25  ((Game Frisbee)
     (Answer (ProvidesScholarshipFor Buffalo ?36))
     (not (Athlete ?36)) (not (PlaysWell ?36 Frisbee)))
```

Interpretation of Clauses As Generic Answers

```
22  ((Answer (ProvidesScholarshipFor Buffalo ?30))
     (not (Sport ?31)) (not (Athlete ?30))
     (not (PlaysWell ?30 ?31)))
```

$$\forall xy[\mathit{Athlete}(x) \land \mathit{Sport}(y) \land \mathit{PlaysWell}(x,y) \\ \Rightarrow \mathit{ProvidesScholarshipFor}(\mathit{Buffalo}, x)]$$

```
24  ((Game ?34) (not (Activity ?34))
     (Answer (ProvidesScholarshipFor Buffalo ?35))
     (not (Athlete ?35)) (not (PlaysWell ?35 ?34)))
```

$$\forall xy[\mathit{Athlete}(x) \land \mathit{Activity}(y) \land \neg \mathit{Game}(y) \land \mathit{PlaysWell}(x,y) \\ \Rightarrow \mathit{ProvidesScholarshipFor}(\mathit{Buffalo}, x)]$$

Interpretation of Clause As Hypothetical Answer

```
25  ((Game Frisbee)
     (Answer (ProvidesScholarshipFor Buffalo ?36))
     (not (Athlete ?36)) (not (PlaysWell ?36 Frisbee)))
```

$$\neg Game(Frisbee) \Rightarrow \forall xy[Athlete(x) \wedge PlaysWell(x, Frisbee)$$
$$\Rightarrow ProvidesScholarshipFor(Buffalo, x)]$$

Rule-Based Systems

Every FOL KB

can be expressed as a set of rules of the form

$\forall \overline{x}(C_1(\overline{x}) \vee \cdots \vee C_m(\overline{x}))$

or

$\forall \overline{x}(A_1(\overline{x}) \wedge \cdots \wedge A_n(\overline{x}) \Rightarrow C_1(\overline{x}) \vee \cdots \vee C_m(\overline{x}))$

or

$\forall \overline{x}(A_1(\overline{x}) \wedge \cdots \wedge A_n(\overline{x}) \Rightarrow C(\overline{x}))$

where $A_i(\overline{x})$ and $C_j(\overline{x})$ are literals.

Wh Questions in Rule-Based Systems

Given rule $\forall\overline{x}(A(\overline{x}) \Rightarrow C(\overline{x}))$

Ask $C(\overline{y})$?

Backchain to subgoal $A(\overline{x})\mu$, where μ is an mgu of $C(\overline{x}))$ and $C(\overline{y}))$

Moral: Unification is generally needed in backward chaining systems.

Unification is correct pattern matching when both structures have variables.

Forward Chaining & Unification

Forward chaining generally matches a ground fact with rule antecedents.

Forward chaining does not generally require unification.

Common Formalizing Difficulties

Every raven is black: $\forall x(Raven(x) \Rightarrow Black(x))$

Some raven is black: $\exists x(Raven(x) \land Black(x))$

Note the satisfying models of the incorrect $\exists x(Raven(x) \Rightarrow Black(x))$

Another Formalizing Difficulty

Note where a Skolem function appears in

$\forall x(Parent(x) \Leftrightarrow \exists y childOf(y, x))$

$$\Leftrightarrow \forall x((Parent(x) \Rightarrow \exists y childOf(y, x)) \\ \wedge((\exists y childOf(y, x)) \Rightarrow Parent(x)))$$

$$\Leftrightarrow \forall x((\neg Parent(x) \vee \exists y childOf(y, x)) \\ \wedge(\neg(\exists y childOf(y, x)) \vee Parent(x)))$$

$$\Leftrightarrow \forall x((\neg Parent(x) \vee \exists y childOf(y, x)) \\ \wedge(\forall y(\neg childOf(y, x)) \vee Parent(x)))$$

$$\Leftrightarrow \forall x(Parent(x) \Rightarrow childOf(f(x), x)) \\ \wedge \forall x \forall y(childOf(y, x) \Rightarrow Parent(x))$$

What’s “First-Order” About FOL?

In a first-order logic:

Predicate and function symbols cannot be arguments of predicates or functions;

Variables cannot be in predicate or function position.

E.G. $\forall r[\mathit{Transitive}(r) \Leftrightarrow \forall xyz[r(x,y) \wedge r(y,z) \Rightarrow r(x,z)]]$
is not a first-order sentence.

“The adjective ’first-order’ is used to distinguish the languages we shall study here from those in which there are predicates having other predicates or functions as arguments or in which predicate quantifiers or function quantifiers are permitted, or both.” [Elliott Mendelson, *Introduction to Mathematical Logic, Fifth Edition,* CRC Press, Boca Raton, FL, 2010, p. 48.]

Russell's Theory of Types

Designed to solve paradox: $\exists s \forall c [s(c) \Leftrightarrow \neg c(c)]$

has instance $S(S) \Leftrightarrow \neg S(S)$

N^{th}-Order Logic

Assign type 0 to individuals and to terms denoting individuals.

Assign type $i+1$ to any set and to any function or predicate symbol that denotes a set, possibly of tuples, such that the maximum type of any of its elements is i.

Also assign type $i+1$ to any variable that range over type i objects.

Note that the type of a functional term is the type of its range—the n^{th} element of the n-tuples of the set which the function denotes.

Syntactically, if the maximum type of the arguments of a ground atomic wff is i, then the type of the predicate is $i+1$.
No predicate of type i may take a ground argument of type i or higher.

First-Order Logic Defined

First-order logic has a language that obeys Russell's Theory of Types, and whose highest type symbol is of type 1.

n^{th}-order logic has a language that obeys Russell's Theory of Types, and whose highest type symbol is of type n.

Ω-ordered logic has no limit, but must still follow the rules.

E.g., $\forall r[\mathit{Transitive}(r) \Leftrightarrow \forall xyz[r(x,y) \land r(y,z) \Rightarrow r(x,z)]]$
is a formula of Second-Order Logic:
Type 0 objects: individuals in the domain
Type 1 symbols: x, y, z because they range over type 0 objects
Type 1 objects: binary relations over type 0 objects
Type 2 symbols: r because it ranges over type 1 objects,
Transitive because it denotes a set of type 1 objects

Nested Beliefs

Can a proposition be an argument of a proposition?

Consider:

$$\begin{array}{l} \forall p(Believes(Solomon, p) \Rightarrow p) \\ Believes(Solomon, Round(Earth)) \Rightarrow Round(Earth) \\ Believes(Solomon, Round(Earth)) \\ \models Round(Earth) \end{array}$$

If $Round(Earth)$ is an atomic wff, it's not a term, and only terms may be arguments of functions and predicates.

Even if it could:

$[\![Round(Earth)]\!] =$ True if $[\![Earth]\!] \in [\![Round]\!]$, else False.

So $[\![Believes(Solomon, Round(Earth))]\!] =$ True
iff $\langle\ [\![Solomon]\!], \textit{True-or-False}\rangle \in [\![Believes]\!]$

Reifying Propositions and the *Holds* Predicate

So how can we represent in FOL
"*Everything that Solomon believes is true*"?

- Reify (some) propositions.
  Make them objects in the domain.
  Represent them using individual constants or functional terms.
- Use $Holds(P)$ to mean
  "*P holds (is true) in the given situation*".
- Examples:

$$\forall p(Believes(Solomon, p) \Rightarrow Holds(p))$$
$$Believes(Solomon, Round(Earth)) \Rightarrow Holds(Round(Earth))$$

Semantics of the *Holds* Predicate

$\forall p(Believes(Solomon, p) \Rightarrow Holds(p)) \wedge Believes(Solomon, Round(Earth))$
$\Rightarrow Holds(Round(Earth))$

Type 0 individuals and terms:
$[Solomon] = [\![Solomon]\!]$ = A person named Solomon
$[Earth] = [\![Earth]\!]$ = The planet Earth
$[Round(Earth)] = [\![Round(Earth)]\!]$ = The proposition that the Earth is round

Type 1 objects and symbols:
p: A variable ranging over type 0 propositions
$[\![Round]\!]$ = A function from type 0 physical objects to type 0 propositions.
$[\![Holds]\!]$ = A set of type 0 propositions.
$[\![Believes]\!]$ = A set of pairs, type 0 People × type 0 propositions

Type 1 atomic formulas:
$[Holds(x)]$ = The type 1 proposition that $[x]$ is True.
$[\![Holds(x)]\!]$ = True if $[\![x]\!] \in [\![Holds]\!]$; False otherwise
$[Believes(x, y)]$ = The type 1 proposition that $[x]$ believes $[y]$
$[\![Believes(x, y)]\!]$ = True if $\langle\ [\![x]\!], [\![y]\!]\ \rangle \in [\![Believes]\!]$; False otherwise

5 Summary of Part I

Artificial Intelligence (AI): A field of computer science and engineering concerned with the computational understanding of what is commonly called intelligent behavior, and with the creation of artifacts that exhibit such behavior.

Knowledge Representation and Reasoning (KR or KRR): A subarea of Artificial Intelligence concerned with understanding, designing, and implementing ways of representing information in computers, and using that information to derive new information based on it.

KR is more concerned with belief than "knowledge". Given that an agent (human or computer) has certain beliefs, what else is reasonable for it to believe, and how is it reasonable for it to act, regardless of whether those beliefs are true and justified.

What is Logic?

Logic is the study of correct reasoning.

There are many systems of logic (logics). Each is specified by specifying:

- Syntax: Specifying what counts as a well-formed expression
- Semantics: Specifying the meaning of well-formed expressions
  - Intensional Semantics: Meaning relative to a Domain
  - Extensional Semantics: Meaning relative to a Situation
- Proof Theory: Defining proof/derivation, and how it can be extended.

Relevance of Logic

Any system that consists of

- a collection of symbol structures,
  well-formed relative to some syntax;
- a set of procedures for adding new structures to that collection based on what's already in there.

is a logic.

But:
Do the symbol structures have a consistent semantics?

Are the procedures sound relative to that semantics?

Soundness is the essence of "correct reasoning."

KR and Logic

Given that a Knowledge Base is represented in a language with a well-defined syntax, a well-defined semantics, and that reasoning over it is a well-defined procedure, a KR system is a logic.

KR research can be seen as a search for the best logic to capture human-level reasoning.

Relations Among Inference Problems

Syntax **Derivation** **Theoremhood**

$$\begin{array}{ccc}
A_1, \ldots, A_n \vdash Q & \Leftrightarrow & \vdash A_1 \wedge \ldots \wedge A_n \Rightarrow Q \\
 & & \\
\Downarrow\Uparrow & & \Downarrow\Uparrow \\
 & & \\
A_1, \ldots, A_n \models Q & \Leftrightarrow & \models A_1 \wedge \ldots \wedge A_n \Rightarrow Q
\end{array}$$

Semantics **Logical Entailment** **Validity**

($\Downarrow$ *Soundness*) ($\Uparrow$ *Completeness*)

Inference/Reasoning Methods

Given a KB/set of assumptions $\mathcal{A}$ and a query $\mathcal{Q}$:

- Model Finding
  - Direct: Find satisfying models of $\mathcal{A}$, see if $\mathcal{Q}$ is True in all of them.
  - Refutation: Find if $\mathcal{A} \cup \{\neg\mathcal{Q}\}$ is unsatisfiable.
- Natural Deduction
  - Direct: Find if $\mathcal{A} \vdash \mathcal{Q}$.
- Resolution
  - Direct: Find if $\mathcal{A} \vdash \mathcal{Q}$ (incomplete).
  - Refutation: Find if $\bigwedge \mathcal{A} \wedge \neg\mathcal{Q}$ is inconsistent.

Classes of Logics

- Propositional Logic
  - Finite number of atomic propositions and models.
  - Model finding and resolution are decision procedures.
- Finite-Model Predicate Logic
  - Finite number of terms, atomic formulae, and models.
  - Reducible to propositional logic.
  - Model finding and resolution are decision procedures.
- First-Order Logic
  - Infinite number of terms, atomic formulae, and models.
  - Not reducible to propositional logic.
  - There are no decision procedures.
  - Resolution plus factoring is refutation complete.

Logics We Studied

1. Standard Propositional Logic
2. Clause Form Propositional Logic
3. Standard Finite-Model Predicate Logic
4. Clause Form Finite-Model Predicate Logic
5. Standard First-Order Predicate Logic
6. Clause Form First-Order Predicate Logic

Proof Procedures We Studied

1. Direct model finding: truth tables, `decreasoner`, `relsat` (complete search) `walksat, gsat` (stochastic search)
2. Semantic tableaux (model-finding refutation)
3. Wang algorithm (model-finding refutation), `wang`
4. Hilbert-style axiomatic (direct), *brief*
5. Fitch-style natural deduction (direct)
6. Resolution (refutation), `prover`, `SNARK`

Utility Notions and Techniques

1. Material implication
2. Possible properties of connectives
    - commutative, associative, idempotent
3. Possible properties of well-formed expressions
    - free, bound variables
    - open, closed, ground expressions
4. Possible semantic properties of wffs
    - contradictory, satisfiable, contingent, valid
5. Possible properties of proof procedures
    - sound, consistent, complete,
    - decision procedure, semi-decision procedure

More Utility Notions and Techniques

5. Substitutions

    application, composition

6. Unification

    most general common instance (mgci),
    most general unifier (mgu)

7. Translation from standard form to clause form

    Conjunctive Normal Form (CNF),
    Skolem functions/constants

8. Resolution Strategies

    subsumption, unit preference, set of support

9. The Answer Literal

Unification

- Unification is a least-commitment method of choosing a substitution for Universal Instantiation ($\forall E$).
- Unification is correct pattern matching when both structures have variables.
- Unification is generally needed in backward chaining systems.

AI-Logic Connections

AI	Logic
rules or domain rules	non-atomic assumptions or non-logical axioms
inference engine procedures	rules of inference
knowledge base	derivation

6 Prolog

6.1 Horn Clauses

A Horn Clause is a clause with at most one positive literal.

Either $\{\neg Q_1(\overline{x}), \ldots, \neg Q_n(\overline{x})\}$ (negative Horn clause)

or $\{C(\overline{x})\}$ (fact or positive or definite Horn clause)

or $\{\neg A_1(\overline{x}), \ldots, \neg A_n(\overline{x}), C(\overline{x})\}$ (positive or definite Horn clause)

which is the same as

$A_1(\overline{x}) \wedge \cdots \wedge A_n(\overline{x}) \Rightarrow C(\overline{x})$

where $A_i(\overline{x})$, $C(\overline{x})$, and $Q(\overline{x})$ are atoms.

SLD Resolution

Selected literals, Linear pattern, over Definite clauses

SLD derivation of clause c from set of clauses S is
$c_1, \ldots, c_n = c$
s.t. $c_1 \in S$
and c_{i+1} is resolvent of c_i and a clause in S. [B&L, p. 87]

If S is a set of Horn clauses,
then there is a resolution derivation of $\{\}$ from S
iff there is an SLD derivation of $\{\}$ from S.

SLDSolve

```
procedure SLDSolve(KB,query) returns true or false {
   /* KB = $\{rule_1, \ldots, rule_n\}$
    * $rule_i = \{h_i, \neg b_{i1}, \ldots, \neg b_{ik_i}\}$
    * query = $\{\neg q_1, \ldots, \neg q_m\}$    */
   if ($m = 0$) return true;
   for $i := 1$ to $n$ {
      if(($\mu :=$ Unify($q_1$, $h_i$)) $\neq$ FAIL
          and SLDSolve(KB, $\{\neg b_{i1}\mu, \ldots, \neg b_{ik_i}\mu, \neg q_2\mu, \ldots, \neg q_m\mu\}$)) {
        return true;
        }
      }
      return false;
}
```

Where h_i, b_{ij}, and q_i are atomic formulae.

See B&L, p. 92

6.2 Prolog

Example Prolog Interaction

```
<timberlake:~/.xemacs:1:35> sicstus
SICStus 4.0.5 (x86_64-linux-glibc2.3): Thu Feb 12 09:48:30 CET 2009
Licensed to SP4cse.buffalo.edu
| ?- consult(user).
% consulting user...
| driver(X) :- drives(X,_).
| passenger(Y) :- drives(_,Y).
| drives(betty,tom).
|
% consulted user in module user, 0 msec 1200 bytes
yes
| ?- driver(X), passenger(Y).
X = betty,
Y = tom ?
yes
| ?- halt.
```

Variables are Capitalized

```
SICStus 4.0.5 (x86-linux-glibc2.3): Thu Feb 12 09:47:39 CET 2009
Licensed to SP4cse.buffalo.edu
| ?- [user].
% compiling user...
| canary(Tweety).
* [Tweety] - singleton variables
|
% compiled user in module user, 10 msec 152 bytes
yes
| ?- canary(Tweety).
true ?
yes
| ?- canary(oscar).
yes
| ?-
```

Prolog Program with Two Answers

```
% From Rich & Knight, 2nd Edition (1991) p. 192.

likesToEat(X,Y) :- cat(X), fish(Y).
cat(X) :- calico(X).
fish(X) :- tuna(X).

tuna(charlie).
tuna(herb).
calico(puss).
```

Listing the Fish Program

```
| ?- listing.
calico(puss).

cat(A) :-
        calico(A).

fish(A) :-
        tuna(A).

likesToEat(A, B) :-
        cat(A),
        fish(B).

tuna(charlie).
tuna(herb).

yes
```

Note: `consult(File)` loads the `File` in interpreted mode, whereas `[File]` loads the `File` in compiled mode. `listing` is only possible in interpreted mode.

Running the Fish Program

```
<timberlake:CSE563:1:39> sicstus
SICStus 4.0.5 (x86_64-linux-glibc2.3): Thu Feb 12 09:48:30 CET 2009
Licensed to SP4cse.buffalo.edu
| ?- ['fish.prolog'].
% compiling /projects/shapiro/CSE563/fish.prolog...
% compiled /projects/shapiro/CSE563/fish.prolog in module user, 0 msec 1808 b
yes

| ?- likesToEat(puss,X).
X = charlie ? ;
X = herb ? ;
no

| ?- halt.
<timberlake:CSE563:1:40>
```

Tracing the Fish Program

```
| ?- ['fish.prolog'].
% consulting /projects/shapiro/CSE563/fish.prolog...
% consulted /projects/shapiro/CSE563/fish.prolog in module user
yes

| ?- trace.
% The debugger will first creep -- showing everything (trace)
yes
% trace
```

Tracing First Answer

```
| ?- likesToEat(puss,X).
        1      1 Call: likesToEat(puss,_442) ?
        2      2 Call: cat(puss) ?
        3      3 Call: calico(puss) ?
        3      3 Exit: calico(puss) ?
        2      2 Exit: cat(puss) ?
        4      2 Call: fish(_442) ?
        5      3 Call: tuna(_442) ?
?       5      3 Exit: tuna(charlie) ?
?       4      2 Exit: fish(charlie) ?
?       1      1 Exit: likesToEat(puss,charlie) ?
X = charlie ? ;
```

Tracing the Second Answer

```
X = charlie ? ;
        1      1 Redo: likesToEat(puss,charlie) ?
        4      2 Redo: fish(charlie) ?
        5      3 Redo: tuna(charlie) ?
        5      3 Exit: tuna(herb) ?
        4      2 Exit: fish(herb) ?
        1      1 Exit: likesToEat(puss,herb) ?
X = herb ? ;
no
% trace
| ?- notrace.
% The debugger is switched off
yes
```

Backtracking Example

Program:

```
bird(tweety).
bird(oscar).
bird(X) :- feathered(X).
feathered(maggie).
large(oscar).
ostrich(X) :- bird(X), large(X).
```

Run (No backtracking needed):

```
| ?- ostrich(oscar).
        1      1 Call: ostrich(oscar) ?
        2      2 Call: bird(oscar) ?
?       2      2 Exit: bird(oscar) ?
        3      2 Call: large(oscar) ?
        3      2 Exit: large(oscar) ?
?       1      1 Exit: ostrich(oscar) ?
yes
```

Backtracking Used

```
| ?- ostrich(X).
        1      1 Call: ostrich(_368) ?
        2      2 Call: bird(_368) ?
?       2      2 Exit: bird(tweety) ?
        3      2 Call: large(tweety) ?
        3      2 Fail: large(tweety) ?
        2      2 Redo: bird(tweety) ?
?       2      2 Exit: bird(oscar) ?
        4      2 Call: large(oscar) ?
        4      2 Exit: large(oscar) ?
?       1      1 Exit: ostrich(oscar) ?
X = oscar ?
yes
```

Backtracking: Effect of Query

`/projects/shapiro/CSE563/Examples/Prolog/backtrack.prolog`:

```
supervisorOf(X,Y) :- managerOf(X,Z), departmentOf(Y,Z).
managerOf(jones,accountingDepartment).
managerOf(smith,itDepartment).
departmentOf(kelly,accountingDepartment).
departmentOf(brown,itDepartment).
```

Backtracking not needed:

```
| ?- supervisorOf(smith,X).
        1      1 Call: supervisorOf(smith,_380) ?
        2      2 Call: managerOf(smith,_772) ?
        2      2 Exit: managerOf(smith,itDepartment) ?
        3      2 Call: departmentOf(_380,itDepartment) ?
        3      2 Exit: departmentOf(brown,itDepartment) ?
        1      1 Exit: supervisorOf(smith,brown) ?
X = brown ?
yes
```

Backtracking Example, part 2

```
supervisorOf(X,Y) :- managerOf(X,Z), departmentOf(Y,Z).
managerOf(jones,accountingDepartment).
managerOf(smith,itDepartment).
departmentOf(kelly,accountingDepartment).
departmentOf(brown,itDepartment).

| ?- supervisorOf(X,brown).
        1      1 Call: supervisorOf(_368,brown) ?
        2      2 Call: managerOf(_368,_772) ?
?       2      2 Exit: managerOf(jones,accountingDepartment) ?
        3      2 Call: departmentOf(brown,accountingDepartment) ?
        3      2 Fail: departmentOf(brown,accountingDepartment) ?
        2      2 Redo: managerOf(jones,accountingDepartment) ?
        2      2 Exit: managerOf(smith,itDepartment) ?
        4      2 Call: departmentOf(brown,itDepartment) ?
        4      2 Exit: departmentOf(brown,itDepartment) ?
        1      1 Exit: supervisorOf(smith,brown) ?
X = smith ?
yes
```

Negation by Failure
& The Closed World Assumption

```
| ?- [user].
% consulting user...
| manager(jones, itSection).
| manager(smith, accountingSection).
|
% consulted user in module user, 0 msec 416 bytes
yes
| ?- manager(smith, itSection).
no
| ?- manager(kelly, accountingSection).
no
```

Negation by failure: “no” means didn’t succeed.

CWA: If it’s not in the KB, it’s not true.

Cut: Preventing Backtracking

KB Without Cut

```
| ?- consult(user).
% consulting user...
| bird(oscar).
| bird(tweety).
| bird(X) :- feathered(X).
| feathered(maggie).
| large(oscar).
| ostrich(X) :- bird(X), large(X).
|
% consulted user in module user, 0 msec 1120 bytes
yes
```

No Backtracking Needed

```
| ?- trace.
% The debugger will first creep -- showing everything (trace)
yes
% trace
| ?- ostrich(oscar).
        1      1 Call: ostrich(oscar) ?
        2      2 Call: bird(oscar) ?
?       2      2 Exit: bird(oscar) ?
        3      2 Call: large(oscar) ?
        3      2 Exit: large(oscar) ?
?       1      1 Exit: ostrich(oscar) ?
yes
% trace
```

Unwanted Backtracking

```
| ?- ostrich(tweety).
        1      1 Call: ostrich(tweety) ?
        2      2 Call: bird(tweety) ?
?       2      2 Exit: bird(tweety) ?
        3      2 Call: large(tweety) ?
        3      2 Fail: large(tweety) ?
        2      2 Redo: bird(tweety) ?
        4      3 Call: feathered(tweety) ?
        4      3 Fail: feathered(tweety) ?
        2      2 Fail: bird(tweety) ?
        1      1 Fail: ostrich(tweety) ?
no
```

No need to try to solve `bird(tweety)` another way.

KB With Cut

```
| ?- consult(user).
% consulting user...
| bird(oscar).
| bird(tweety).
| bird(X) :- feathered(X).
| feathered(maggie).
| large(oscar).
| ostrich(X) :- bird(X), !, large(X).
|
% consulted user in module user, 0 msec -40 bytes
yes
% trace
```

No Extra Backtracking

```
| ?- ostrich(tweety).
        1      1 Call: ostrich(tweety) ?
        2      2 Call: bird(tweety) ?
?       2      2 Exit: bird(tweety) ?
        3      2 Call: large(tweety) ?
        3      2 Fail: large(tweety) ?
        1      1 Fail: ostrich(tweety) ?
no
% trace
```

Cut Fails the Head Instance: Program

```
| ?- [user].
% compiling user...
| yellow(bigbird).
| bird(tweety).
| canary(X) :- bird(X), !, yellow(X).
| canary(X).
* [X] - singleton variables
|
% compiled user in module user, 0 msec 600 bytes
yes
| ?- canary(fred).
yes
| ?- canary(bigbird).
yes
| ?- canary(tweety).
no
| ?-
```

`fail`: Forcing Failure

If something is a canary, it is not a penguin.

```
| ?- consult(user).
% consulting user...
| penguin(X) :- canary(X), !, fail.
| canary(tweety).
|
% consulted user in module user, 0 msec 416 bytes
yes
% trace
| ?- penguin(tweety).
        1      1 Call: penguin(tweety) ?
        2      2 Call: canary(tweety) ?
        2      2 Exit: canary(tweety) ?
        1      1 Fail: penguin(tweety) ?
no
% trace
```

Cut Fails the Head Instance: Program

```
penguin(X) :- canary(X), !, fail.
penguin(X) :- bird(X), swims(X).

canary(tweety).
bird(willy).
swims(willy).
```

Cut Fails the Head Instance: Run

```
| ?- penguin(willy).
        1      1 Call: penguin(willy) ?
        2      2 Call: canary(willy) ?
        2      2 Fail: canary(willy) ?
        3      2 Call: bird(willy) ?
        3      2 Exit: bird(willy) ?
        4      2 Call: swims(willy) ?
        4      2 Exit: swims(willy) ?
        1      1 Exit: penguin(willy) ?
yes
% trace
| ?- penguin(tweety).
        1      1 Call: penguin(tweety) ?
        2      2 Call: canary(tweety) ?
        2      2 Exit: canary(tweety) ?
        1      1 Fail: penguin(tweety) ?
no
```

Cut Fails Head Alternatives

```
  | ?- penguin(X).
        1      1 Call: penguin(_368) ?
        2      2 Call: canary(_368) ?
        2      2 Exit: canary(tweety) ?
        1      1 Fail: penguin(_368) ?
no
```

Moral:
Use cut when seeing if a ground atom is satisfied (T/F question),
but not when generating satisfying instances (wh questions).

Bad Rule Order

```
penguin(X) :- bird(X), swims(X).
penguin(X) :- canary(X), !, fail.
bird(X) :- canary(X).
canary(tweety).

% trace
| ?- penguin(tweety).
        1      1 Call: penguin(tweety) ?
        2      2 Call: bird(tweety) ?
        3      3 Call: canary(tweety) ?
        3      3 Exit: canary(tweety) ?
        2      2 Exit: bird(tweety) ?
        4      2 Call: swims(tweety) ?
        4      2 Fail: swims(tweety) ?
        5      2 Call: canary(tweety) ?
        5      2 Exit: canary(tweety) ?
        1      1 Fail: penguin(tweety) ?
no
```

Good Rule Order

```
penguin(X) :- canary(X), !, fail.
penguin(X) :- bird(X), swims(X).
bird(X) :- canary(X).
canary(tweety).

% trace
| ?- penguin(tweety).
        1      1 Call: penguin(tweety) ?
        2      2 Call: canary(tweety) ?
        2      2 Exit: canary(tweety) ?
        1      1 Fail: penguin(tweety) ?
no
```

SICSTUS Allows “or” In Body.

```
bird(willy).
swims(willy).
canary(tweety).
penguin(X) :-
        canary(X), !, fail;
        bird(X), swims(X).
bird(X) :- canary(X).

| ?- ['twoRuleCutOr.prolog'].
% compiling /projects/shapiro/CSE563/twoRuleCutOr.prolog...
* clauses for user:bird/1 are not together
* Approximate lines: 8-10, file: '/projects/shapiro/CSE563/twoRuleCutOr.prolog'
% compiled /projects/shapiro/CSE563/twoRuleCutOr.prolog in module user, 0 msec 928 bytes
yes
| ?- penguin(willy).
yes
| ?- penguin(tweety).
no
```

`not`: “Negated” Antecedents

A bird that is not a canary is a penguin.

```
| penguin(X) :- bird(X), !, \+canary(X).
| bird(opus).
| canary(tweety).
% compiled user in module user, 0 msec 512 bytes

| ?- penguin(opus).
        1      1 Call: penguin(opus) ?
        2      2 Call: bird(opus) ?
        2      2 Exit: bird(opus) ?
        3      2 Call: canary(opus) ?
        3      2 Fail: canary(opus) ?
        1      1 Exit: penguin(opus) ?
yes
```

`\+` is SICStus Prolog’s version of `not`.

It is negation by failure, not logical negation.

Can Use Functions

```
driver(X) :- drives(X,_).
drives(mother(X),X) :- schoolchild(X).
schoolchild(betty).
schoolchild(tom).

| ?- driver(X).
X = mother(betty) ? ;
X = mother(tom) ? ;
no
```

Infinitely Growing Terms

```
driver(X) :- drives(X,_).
drives(mother(X),X) :- commuter(X).
commuter(betty).
commuter(tom).
commuter(mother(X)) :- commuter(X).

| ?- driver(X).
X = mother(betty) ? ;
X = mother(tom) ? ;
X = mother(mother(betty)) ? ;
X = mother(mother(tom)) ? ;
X = mother(mother(mother(betty))) ? ;
X = mother(mother(mother(tom))) ?
yes
```

Prolog Does Not Do the Occurs Check

```
<pollux:CSE563:2:31> sicstus
...
| ?- [user].
% consulting user...
| mother(motherOf(X), X).
|
% consulted user in module user, 0 msec 248 bytes
yes
| ?- mother(Y, Y).
Y = motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(
             motherOf(motherOf(motherOf(motherOf(...))))))))))
yes
| ?-
```

“=” and “is”

```
| ?- p(X, b, f(c,Y)) = p(a, U, f(V,U)).
U = b,
V = c,
X = a,
Y = b ?
yes
| ?- X is 2*(3+6).
X = 18 ?
yes
| ?- X = 2*(3+6).
X = 2*(3+6) ?
yes
| ?- X is 2*(3+6), Y is X/3.
X = 18,
Y = 6.0 ?
yes
| ?- Y is X/3, X is 2*(3+6).
! Instantiation error in argument 2 of is/2
! goal:  _76 is _73/3
```

Avoid Left Recursive Rules

To define `ancestor` as the transitive closure of `parent`.

The base case: `ancestor(X,Y) :- parent(X,Y).`

Three possible recursive cases:

1. `ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).`
2. `ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).`
3. `ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).`

Versions (2) and (3) will cause infinite loops.

7 A Potpourri of Subdomains

Taxonomies: Categories as Intensional Sets

In mathematics, a set is defined by its members.

This is an **extensional set**.

Plato: *Man is a featherless biped.*

An **intensional** set is defined by properties.

Aristotle: *Man is a rational animal.*

A category (type, class) is an intensional set.

Taxonomies: Need for Two Relations

With sets, there's a difference between

set membership, $\in$ $5 \in \{1,3,5,7,9\}$

and subset, $\subset, \subseteq$ $\{1,3,5,7,9\} \subset \{1,2,3,4,5,6,7,8,9\}$

One difference is that subset is transitive:
$\{1,3,5\} \subset \{1,3,5,7,9\}$ and $\{1,3,5,7,9\} \subset \{1,2,3,4,5,6,7,8,9\}$
and $\{1,3,5\} \subset \{1,2,3,4,5,6,7,8,9\}$

membership is not:
$5 \in \{1,3,5,7,9\}$ and $\{1,3,5,7,9\} \in \{\{1,3,5,7,9\},\{2,4,6,8\}\}$
but $5 \notin \{\{1,3,5,7,9\},\{2,4,6,8\}\}$

Similarly, we need both the instance relation and the subcategory relation.

Taxonomies:
Categories as Unary Predicates

One way to represent taxonomies:

$Canary(Tweety)$

$\forall x[(Canary(x) \Rightarrow Bird(x)]$
$\forall x[(Bird(x) \Rightarrow Vertebrate(x)]$
$\forall x[(Vertebrate(x) \Rightarrow Chordate(x)]$
$\forall x[(Chordate(x) \Rightarrow Animal(x)]$

Taxonomies: Reifying

To reify: to make a thing of.
Allows discussion of “predicates” in FOL.

Membership: *Member* or *Instance* or *Isa*
$Isa(Tweety, Canary)$

Subcategory: *Subclass* or *Ako* (sometimes, even, *Isa*)
$Ako(Canary, Bird)$
$Ako(Bird, Vertebrate)$
$Ako(Vertebrate, Chordate)$
$Ako(Chordate, Animal)$

Axioms:
$\forall x \forall c_1 \forall c_2 [Isa(x, c_1) \wedge Ako(c_1, c_2) \Rightarrow Isa(x, c_2)]$
$\forall c_1 \forall c_2 \forall c_3 [Ako(c_1, c_2) \wedge Ako(c_2, c_3) \Rightarrow Ako(c_1, c_3)]$

Discussing Categories

$$Isa(Canary, Species)$$

$$Isa(Bird, Class)$$

$$Isa(Chordate, Phylum)$$

$$Isa(Animal, Kingdom)$$

$$Extinct(Dinosaur)$$

Note: That's *Isa*, not *Ako*.

If categories are predicates, requires second-order logic.

Other relationships: exhaustive subcategories, disjoint categories, partitions.

DAG (directed acyclic graph), rather than just a tree.
E.g., human: man vs. woman; child vs. adult vs. senior.

Transitive Closure

It's sometimes useful (especially in Prolog)
to have a second relation, R_2
be the transitive closure of a relation, R_1.

$$
\begin{aligned}
&\forall R_1, R_2[transitiveClosureOf(R_2, R_1)\\
&\qquad\qquad \Leftrightarrow [\forall x, y(R_1(x, y) \Rightarrow R_2(x, y))\\
&\qquad\qquad\quad \wedge \forall x, y, z[R_1(x, y) \wedge R_2(y, z) \Rightarrow R_2(x, z)]]
\end{aligned}
$$

E.g. *ancestor* is the transitive closure of *parent*:

$$\forall x, y[parent(x, y) \Rightarrow ancestor(x, y)]$$

$$\forall x, y, z[parent(x, y) \wedge ancestor(y, z) \Rightarrow ancestor(x, z)]$$

7.2 Time

How would you represent time?

Discuss

Subjective *vs.* Objective: Subjective

Make now an individual in the domain.

Include other times relative to now.

OK if time doesn't move.

Subjective *vs.* Objective: Objective

Make `now` a meta-logical variable with some time-denoting term as value.

Relate times to each other, *e.g.* $\mathit{Before}(\mathit{t1}, \mathit{t2})$.

Now can move by giving `now` a new value.

Points *vs.* Intervals: Points

Use numbers: integers, rationals, reals?

Computer reals aren't really dense.

How to assign numbers to times?

Granularity: How big, numerically, is a day, or any other interval of time?

If an interval is defined as a pair of points, which interval is the midpoint in, if one interval immediately follows another?

Points *vs.* Intervals: Intervals

Use intervals only: no points at all.

More cognitively accurate.

Granularity is not fixed.

A “point” is just an interval with nothing inside it.

James Allen’s Interval Relations

```
               x                              x
before(x,y)  |---|  |---|      meets(x,y)   |---|---|
                      y                           y
                                             x
                   x                       |---|
overlaps(x,y)   |----|         equals(x,y) |---|
                  |----|                     y
                     y
                   x                           x
starts(x,y)      |---|      finishes(x,y)    |---|
                 |-----|                   |-----|
                    y                         y
                  x
during(x,y)     |---|
               |-----|
                  y
```

[James F. Allen, Maintaining Knowledge About Temporal Intervals, *Communications of the ACM 26,* 11 (Nov 1983), 832–843.]

A Smaller Set of Temporal Relations

If fewer distinctions are needed, one may use

$\mathit{before}(x, y)$ for Allen's $\mathit{before}(x, y) \vee \mathit{meets}(x, y)$

$\mathit{during}(x, y)$ for Allen's $\mathit{starts}(x, y) \vee \mathit{during}(x, y) \vee \mathit{finishes}(x, y)$

$\mathit{overlaps}(x, y)$ and $\mathit{equals}(x, y)$

and appropriate converses.

7.3 Things *vs.* Substances
Count Nouns *vs.* Mass Nouns

A count noun denotes a thing.
Count nouns can be singular or plural.
Things can be counted.
One dog. Two dogs.

A mass noun denotes a substance.
Mass nouns can only be singular.
One can have a quantity of a substance.
A glass of water. A pint of ice cream.

A Quantity of a Substance is a Thing

water a substance

a lake = a body of water a thing

lakes a plurality of things

40 acres of lakes a quantity of a substance

Nouns with mass and count senses

A noun might have both senses.

a piece of pie vs. *A piece of a pie*

two pieces of steak vs. *two steaks*

Any count noun can be "massified".
Any thing can be put through "the universal grinder".
I can't get up; I've got cat on my lap.

8 SNePS

8.1 SNePSLOG Semantics
The Intensional Domain of (Mental) Entities

Frege: *The Morning Star is the Evening Star.*
different from *The Morning Star is the Morning Star.*

Russell: *George IV wanted to know whether Scott was the author of Waverly.*
not *George IV wanted to know whether the author of Waverly was the author of Waverly.*

Jerry Siegel and Joe Shuster: Clark Kent is a mild-mannered reporter; Superman is the man of steel.

Intensions vs. Extensions

the Morning Star and *the Evening Star*
Scott and *the author of Waverly*
Clark Kent and *Superman*

are different intensions, or intensional entities, or mental entities, or just entities,

even though they are coreferential, or extensionally equivalent, or have the same extensions.

SNePSLOG Semantics
Intensional Representation

SNePSLOG individual ground terms denote intensions, (mental) entities.

Mental entities include propositions.

Propositions are first-class members of the domain.

SNePSLOG wffs denote propositions.

Assume that for every entity in the domain there is a term that denotes it.

Make unique names assumption: no two terms denote the same entity.

The Knowledge Base

Think of the SNePS KB as the contents of the mind of an intelligent agent.

The terms in the KB denote mental entities that the agent has conceived of (so far).

Some of the wffs are **asserted**.
These denote propositions that the agent believes.

The rules of inference sanction believing some additional proposition(s), but drawing that inference is optional.
I.e., the agent is not logically omniscient.

8.2 SNePSLOG Syntax
Atomic Symbols

Individual Constants, Variables, Function Symbols:

any Lisp symbol, number, or string.

All that matters is the sequence of characters.
I.e. `"4"`, `\4`, and `4`, are the same.

The sets of individual constants, variables, and function symbols should be distinct, but don't have to be.

SNePSLOG Syntax
Terms

An individual constant is a term.

A variable is a term.

If `t1, ..., tn` are terms, then {`t1, ..., tn`} is a set of terms.

If `f` is a function symbol or a variable, then `f()` is a term.

If `t1, ..., tn` are terms or sets of terms and `f` is a function symbol or variable, then `f(t1, ..., tn)` is a term.

A function symbol needn't have a fixed arity, but it might be a mistake of formalization otherwise.

SNePSLOG Syntax
Atomic Wffs

If `x` is a variable, then `x` is a wff.
If `P` is a proposition-valued function symbol or variable,
then `P()` is a wff.

If `t1, ..., tn` are terms or sets of terms
and `P` is a proposition-valued function symbol or variable,
then `P(t1, ..., tn)` is a wff.

A predicate symbol needn't have a fixed arity, but it might be a mistake of formalization otherwise.

If P1, . . . , Pn are wffs, then {P1, . . . , Pn} is a set of wffs.

Abbreviation: If P is a wff, then P is an abbreviation of {P}.

Every wff is a proposition-denoting term.

SNePSLOG Syntax/Semantics
AndOr

If $\{P_1, \ldots, P_n\}$ is a set of wffs (proposition-denoting terms), and i and j are integers such that $0 <= i <= j <= n$, then

`andor(`i, j`)`$\{P_1, \ldots, P_n\}$ is a wff (proposition-denoting term).

The proposition that at least i and at most j of $P_1, \ldots, P_n$ are True.

SNePSLOG Syntax/Semantics
Abbreviations of AndOr

`~`P = `andor(0,0)`$\{P\}$

`and`$\{P_1, \ldots, P_n\}$ = `andor(`n, n`)`$\{P_1, \ldots, P_n\}$
`or`$\{P_1, \ldots, P_n\}$ = `andor(`$1, n$`)`$\{P_1, \ldots, P_n\}$
`nand`$\{P_1, \ldots, P_n\}$ = `andor(`$0, n-1$`)`$\{P_1, \ldots, P_n\}$
`nor`$\{P_1, \ldots, P_n\}$ = `andor(`$0, 0$`)`$\{P_1, \ldots, P_n\}$
`xor`$\{P_1, \ldots, P_n\}$ = `andor(`$1, 1$`)`$\{P_1, \ldots, P_n\}$

P_1 `and ...  and` P_n = `andor(`n, n`)`$\{P_1, \ldots, P_n\}$
P_1 `or ...  or` P_n = `andor(`$1, n$`)`$\{P_1, \ldots, P_n\}$

SNePSLOG Syntax/Semantics
Thresh

If $\{P_1, \ldots, P_n\}$ is a set of wffs (proposition-denoting terms) and i and j are integers such that $0 <= i <= j <= n$, then

`thresh`$(i, j)\{P_1, \ldots, P_n\}$ is a wff (proposition-denoting term).

The proposition that
either fewer than i or more than j of $P_1, \ldots, P_n$ are True.

SNePSLOG Syntax/Semantics
Abbreviations of Thresh

`iff`$\{P_1, \ldots, P_n\}$
is an abbreviation of `thresh(`$1, n-1$`)`$\{P_1, \ldots, P_n\}$

P_1 `<=>` $\cdots$ `<=>` P_n
is an abbreviation of `thresh(`$1, n-1$`)`$\{P_1, \ldots, P_n\}$

`thresh(`i`)`$\{P_1, \ldots, P_n\}$
is an abbreviation of `thresh(`$i, n-1$`)`$\{P_1, \ldots, P_n\}$

SNePSLOG Syntax/Semantics
Numerical Entailment

If $\{P_1, \ldots, P_n\}$ and $\{Q_1, \ldots, Q_m\}$ are sets of wffs (proposition-denoting terms), and i is an integer, $1 <= i <= n$, then
$\{P_1, \ldots, P_n\}$ i`=>` $\{Q_1, \ldots, Q_m\}$ is a wff (proposition-denoting term).

The proposition that whenever at least i of $P_1, \ldots, P_n$ are True, then so is any $Q_j \in \{Q_1, \ldots, Q_m\}$.

SNePSLOG Syntax/Semantics
Abbreviations of Numerical Entailment

$\{P_1, \ldots, P_n\}$ `=>` $\{Q1, \ldots, Qm\}$
is an abbreviation of $\{P_1, \ldots, P_n\}$ `1=>` $\{Q_1, \ldots, Q_m\}$

$\{P_1, \ldots, P_n\}$ `v=>` $\{Q1, \ldots, Qm\}$
is also an abbreviation of $\{P_1, \ldots, P_n\}$ `1=>` $\{Q_1, \ldots, Q_m\}$

$\{P_1, \ldots, P_n\}$ `&=>` $\{Q1, \ldots, Qm\}$
is an abbreviation of $\{P_1, \ldots, P_n\}$ n`=>` $\{Q_1, \ldots, Q_m\}$

SNePSLOG Syntax/Semantics
Universal Quantifier

If P is a wff (proposition-denoting term)
and $x_1, \ldots, x_n$ are variables, then

`all(`$x_1, \ldots, x_n$`)(`P`)` is a wff (proposition-denoting term).

The proposition that for every sequence of ground terms, $t_1, \ldots, t_n$, P$\{t_1/x_1, \ldots, t_n/x_n\}$ is True.

SNePSLOG Syntax/Semantics
Numerical Quantifier

If $\mathcal{P}$ and $\mathcal{Q}$ are sets of wffs, $x_1, \ldots, x_n$ are variables, and i, j, and k are integers such that $0 <= i <= j <= k$, then

`nexists(`i, j, k`)(`$x_1, \ldots, x_n$`)(`$\mathcal{P}$`:` $\mathcal{Q}$`)` is a wff.

The proposition that there are k sequences of ground terms, $t_1, \ldots, t_n$, that satisfy every $P \in \mathcal{P}$, and, of them, at least i and at most j also satisfy every $Q \in \mathcal{Q}$.

SNePSLOG Syntax/Semantics
Abbreviations of Numerical Quantifier

`nexists(_, `j`, _)(`$x_1, \ldots, x_n$`)(`$\mathcal{P}$`:  `$\mathcal{Q}$`)`
is an abbreviation of `nexists(`$0, j, \infty$`)(`$x_1, \ldots, x_n$`)(`$\mathcal{P}$`:  `$\mathcal{Q}$`)`

`nexists(`i`, _, `k`)(`$x_1, \ldots, x_n$`)(`$\mathcal{P}$`:  `$\mathcal{Q}$`)`
is an abbreviation of `nexists(`i, k, k`)(`$x_1, \ldots, x_n$`)(`$\mathcal{P}$`:  `$\mathcal{Q}$`)`

SNePSLOG Syntax/Semantics
Wffs are Terms

Every wff is a proposition-denoting term.

So, e.g., `Believes(Tom, ~Penguin(Tweety))`
is a wff, and a well-formed term.

For a more complete, more formal syntax, see
The *SNePS 2.7.1 User's Manual,*
`http://www.cse.buffalo.edu/sneps/Manuals/manual271.pdf`.

8.3 SNePSLOG Proof Theory
Implemented Rules of Inference
Reduction Inference

Reduction Inference$_1$: If α is a set of terms and $\beta \subset \alpha$,

$\mathtt{P(t}_1, \ldots, \alpha\mathtt{,\ ...t}_n\mathtt{)} \vdash \mathtt{P(t}_1, \ldots, \beta, \ldots\ \mathtt{t}_n\mathtt{)}$

Reduction Inference$_2$: If α is a set of terms, and `t` $\in \alpha$,

$\mathtt{P(t}_1, \ldots, \alpha\mathtt{,\ ...t}_n\mathtt{)} \vdash \mathtt{P(t}_1\mathtt{,\ ...,\ t,\ ...t}_n\mathtt{)}$

Example of Reduction Inference

```
: clearkb
Knowledge Base Cleared

: Member({Fido, Rover, Lassie}, {dog, pet}).
  wff1!:  Member({Lassie,Rover,Fido},{pet,dog})
 CPU time : 0.00

: Member ({Fido, Lassie}, dog)?
  wff2!:  Member({Lassie,Fido},dog)
 CPU time : 0.00
```

SNePSLOG Proof Theory
Implemented Rules of Inference
for AndOr

AndOr $\mathbf{I}_1$: $\texttt{P}_1\texttt{, ..., P}_n \vdash \texttt{andor}(n,n)\{\texttt{P}_1\texttt{, ..., P}_n\}$

AndOr $\mathbf{I}_2$: $\texttt{\~{}P}_1\texttt{, ..., \~{}P}_n \vdash \texttt{andor(0,0)}\{\texttt{P}_1\texttt{, ..., P}_n\}$

AndOr $\mathbf{E}_1$: $\texttt{andor}(i,j)\{\texttt{P}_1\texttt{, ..., P}_n\}, \texttt{\~{}P}_1\texttt{, ..., \~{}P}_{n-i} \vdash \texttt{P}_j$
for $n-i < j \leq n$

AndOr $\mathbf{E}_2$: $\texttt{andor}(i,j)\{\texttt{P}_1\texttt{, ..., P}_n\}\texttt{, P}_1\texttt{, ..., P}_j \vdash \texttt{\~{}P}_k,$
for $j < k \leq n$

SNePSLOG Proof Theory Implemented Rules of Inference for Thresh

Thresh $\mathbf{E_1}$: When at least i args are true, and at least $n-j-1$ args are false, conclude that any other arg is true.

$\texttt{thresh(}i,j\texttt{)}\{\texttt{P}_1\texttt{, ..., P}_n\}\texttt{,}$

$\texttt{P}_1\texttt{, ..., P}_i\texttt{, }\neg\texttt{P}_{i+1}\texttt{, ..., }\neg\texttt{P}_{i+n-j-1}$

$\vdash \texttt{P}_{i+n-j}$

Thresh $\mathbf{E_2}$: When at least $i-1$ args are true, and at least $n-j$ args are false, conclude that any other arg is false.

$\texttt{thresh(}i,j\texttt{)}\{\texttt{P}_1\texttt{, ..., P}_n\}\texttt{,}$

$\texttt{P}_1\texttt{, ..., P}_{i-1}\texttt{, }\neg\texttt{P}_{i+1}\texttt{, ..., }\neg\texttt{P}_{i+n-j}$

$\vdash \neg\texttt{P}_i$

SNePSLOG Proof Theory
Implemented Rules of Inference
for `&=>`

`&=>I`: If $\mathcal{A}$, $\mathtt{P}_1$, ..., $\mathtt{P}_n \vdash \mathtt{Q}_i$ for $1 \leq i \leq m$
then $\mathcal{A} \vdash \{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$ `&=>` $\{\mathtt{Q}_1, \ldots, \mathtt{Q}_m\}$

`&=>E`: $\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$ `&=>` $\{\mathtt{Q}_1, \ldots, \mathtt{Q}_m\}$, $\mathtt{P}_1$, ..., $\mathtt{P}_n \vdash \mathtt{Q}_i$,
for $1 \leq i \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for `v=>`

`v=>I:` If $\mathcal{A} \vdash$ `P v=> Q` and $\mathcal{A} \vdash$ `Q v=> R` then $\mathcal{A} \vdash$ `P v=> R`

`v=>E:` $\{\mathtt{P}_1\mathtt{,\ \ldots,\ P}_n\}$ `v=>` $\{\mathtt{Q}_1\mathtt{,\ \ldots,\ Q}_m\}$, $\mathtt{P}_i, \vdash \mathtt{Q}_j$,
for $1 \leq i \leq n, 1 \leq j \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for `i=>`

`i=>E:` $\{$`P`$_1$`, ..., P`$_n\}$ **`i=>`** $\{$`Q`$_1$`, ..., Q`$_m\}$`, P`$_1$`, ..., P`$_i \vdash$ `Q`$_j$,
for $1 \leq i \leq n, 1 \leq j \leq m$

SNePSLOG Proof Theory
Implemented Rules of Inference
for `all`

Universal Elimination for universally quantified versions of `andor`, `thresh`, `v=>`, `&=>`, and *i*`=>`.

UVBR & Symmetric Relations

In any substitution $\{t_1/x_1, \ldots, t_n/x_n\}$, if $x_i \neq x_j$, then $t_i \neq t_j$

```
: all(u,v,x,y)(childOf({u,v}, {x,y}) => Siblings({u,v})).

: childOf({Tom,Betty,John,Mary}, {Pat,Harry}).

: Siblings({?x,?y})?

  wff14!:  Siblings({Mary,John})
  wff13!:  Siblings({John,Betty})
  wff12!:  Siblings({Betty,Tom})
  wff11!:  Siblings({Mary,Betty})
  wff10!:  Siblings({John,Tom})
  wff9!:  Siblings({Mary,Tom})
```

SNePSLOG Proof Theory
Implemented Rules of Inference
for `nexists`

nexists $\mathbf{E}_1$:

$\mathtt{nexists}(\mathtt{i},\mathtt{j},\mathtt{k})(\overline{\mathtt{x}})(\overline{\mathtt{P}}(\overline{\mathtt{x}}):\mathtt{Q}(\overline{\mathtt{x}})),$

$\overline{\mathtt{P}}(\overline{\mathtt{t}}_1),\mathtt{Q}(\overline{\mathtt{t}}_1),\ \ldots,\ \overline{\mathtt{P}}(\overline{\mathtt{t}}_\mathtt{j}),\ Q(\bar{t}_j),$

$\overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{j}+1})$

$\vdash \neg\mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{j}+1})$

nexists $\mathbf{E}_2$:

$\mathtt{nexists}(\mathtt{i},\mathtt{j},\mathtt{k})(\overline{\mathtt{x}})(\overline{\mathtt{P}}(\overline{\mathtt{x}}):\mathtt{Q}(\overline{\mathtt{x}})),$

$\overline{\mathtt{P}}(\overline{\mathtt{t}}_1),\neg\mathtt{Q}(\overline{\mathtt{t}}_1),\ldots,\overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{k}-\mathtt{i}}),\neg\mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{k}-\mathtt{i}}),$

$\overline{\mathtt{P}}(\overline{\mathtt{t}}_{\mathtt{k}-\mathtt{i}+1})$

$\vdash \mathtt{Q}(\overline{\mathtt{t}}_{\mathtt{k}-\mathtt{i}+1})$

8.4 Loading SNePSLOG

```
cl-user(2): :ld /projects/snwiz/bin/sneps
; Loading /projects/snwiz/bin/sneps.lisp
;;; Installing streamc patch, version 2.
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.7 [PL:2 2010/09/04 02:35:18] loaded.
Type `(sneps)' or `(snepslog)' to get started.

cl-user(3): (snepslog)


   Welcome to SNePSLOG (A logic interface to SNePS)

Copyright (C) 1984--2010 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type `copyright' for detailed copyright information.
Type `demo' for a list of example applications.
```

Running SNePSLOG

```
cl-user(3): (snepslog)

   Welcome to SNePSLOG (A logic interface to SNePS)

Copyright (C) 1984--2010 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type ‘copyright’ for detailed copyright information.
Type ‘demo’ for a list of example applications.

: clearkb
Knowledge Base Cleared
 CPU time : 0.00

: Member({Fido, Rover, Lassie}, {dog, pet}).
  wff1!:  Member({Lassie,Rover,Fido},{pet,dog})
 CPU time : 0.00

: Member ({Fido, Lassie}, dog)?
  wff2!:  Member({Lassie,Fido},dog)
 CPU time : 0.00
```

Common SNePSLOG Commands

```
: clearkb
Knowledge Base Cleared

: all(x)(dog(x) => animal(x)). ; Assert into the KB
  wff1!:  all(x)(dog(x) => animal(x))

: dog(Fido). ; Assert into the KB
  wff2!:  dog(Fido)

: dog(Fido)?? ; Query assertion without inference
  wff2!:  dog(Fido)
```

Common SNePSLOG Commands

```
: animal(Fido)?? ; Query assertion without inference

: animal(Fido)? ; Query assertion with inference
  wff3!:  animal(Fido)

: dog(Rover)! ; Assert into the KB & do forward inference
  wff6!:  animal(Rover)
  wff5!:  dog(Rover)

: list-asserted-wffs ; Print all asserted wffs
  wff6!:  animal(Rover)
  wff5!:  dog(Rover)
  wff3!:  animal(Fido)
  wff2!:  dog(Fido)
  wff1!:  all(x)(dog(x) => animal(x))
```

Tracing Inference

```
: trace inference
Tracing inference.

: animal(Fido)?

I wonder if  wff3:   animal(Fido)
holds within the BS defined by context default-defaultct

I wonder if  wff5:   dog(Fido)
holds within the BS defined by context default-defaultct

I know  wff2!:  dog({Rover,Fido})

Since  wff1!:  all(x)(dog(x) => animal(x))
and  wff5!:  dog(Fido)
I infer  wff3:   animal(Fido)

  wff3!:  animal(Fido)
 CPU time : 0.01

: untrace inference
Untracing inference.
 CPU time : 0.00

: animal(Rover)?
  wff6!:  animal(Rover)
```

Recursive Rules
Don't Cause Infinite Loops

```
: all(x,y)(parentOf(x,y) => ancestorOf(x,y)).
  wff1!:  all(y,x)(parentOf(x,y) => ancestorOf(x,y))

: all(x,y,z)({ancestorOf(x,y), ancestorOf(y,z)} &=> ancestorOf(x,z)).
  wff2!:  all(z,y,x)({ancestorOf(y,z),ancestorOf(x,y)} &=> {ancestorOf(x,z)})

: parentOf(Sam,Lou).
  wff3!:  parentOf(Sam,Lou)

: parentOf(Lou,Stu).
  wff4!:  parentOf(Lou,Stu)

: ancestorOf(Max,Stu).
  wff5!:  ancestorOf(Max,Stu)

: ancestorOf(?x,Stu)?
  wff8!:  ancestorOf(Sam,Stu)
  wff6!:  ancestorOf(Lou,Stu)
  wff5!:  ancestorOf(Max,Stu)
 CPU time : 0.01
```

Infinitely Growing Terms
Get Cut Off

```
: all(x)(Duck(motherOf(x)) => Duck(x)).
  wff1!:  all(x)(Duck(motherOf(x)) => Duck(x))
 CPU time : 0.00

: Duck(Daffy)?

I wonder if  wff2:   Duck(Daffy)
holds within the BS defined by context default-defaultct

I wonder if  wff5:   Duck(motherOf(Daffy))
holds within the BS defined by context default-defaultct

I wonder if  wff8:   Duck(motherOf(motherOf(Daffy)))
holds within the BS defined by context default-defaultct
...
I wonder if  wff32:   Duck(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(Daffy))
holds within the BS defined by context default-defaultct
SNIP depth cutoff beyond *depthcutoffback* = 10

I wonder if  wff35:   Duck(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(motherOf(mother
holds within the BS defined by context default-defaultct
SNIP depth cutoff beyond *depthcutoffback* = 10
SNIP depth cutoff beyond *depthcutoffback* = 10
 CPU time : 0.05
```

Eager-Beaver Search

```
: all(x)(Duck(motherOf(x)) => Duck(x)).
  wff1!:  all(x)(Duck(motherOf(x)) => Duck(x))
: all(x)({walksLikeaDuck(x), talksLikeaDuck(x)} &=> Duck(x)).
  wff2!:  all(x)({talksLikeaDuck(x),walksLikeaDuck(x)} &=> {Duck(x)})
: and{talksLikeaDuck(Daffy),walksLikeaDuck(Daffy)}.
  wff5!:  walksLikeaDuck(Daffy) and talksLikeaDuck(Daffy)

: Duck(Daffy)? (1)
I wonder if  wff6:   Duck(Daffy)

I wonder if  wff9:   Duck(motherOf(Daffy))

I wonder if  wff3:   talksLikeaDuck(Daffy)

I wonder if  wff4:   walksLikeaDuck(Daffy)

It is the case that  wff4:   walksLikeaDuck(Daffy)

It is the case that  wff3:   talksLikeaDuck(Daffy)

Since  wff2!:  all(x)({talksLikeaDuck(x),walksLikeaDuck(x)} &=> {Duck(x)})
and  wff3!:  talksLikeaDuck(Daffy)
and  wff4!:  walksLikeaDuck(Daffy)
I infer  wff6:   Duck(Daffy)

  wff6!:  Duck(Daffy)
 CPU time : 0.02
```

Contradictions

The KB

```
: clearkb
Knowledge Base Cleared

: all(x)(nand{Mammal(x), Fish(x)}).
  wff1!:  all(x)(nand{Fish(x),Mammal(x)})

: all(x)(LivesInWater(x) => Fish(x)).
  wff2!:  all(x)(LivesInWater(x) => Fish(x))

: all(x)(BearsYoungAlive(x) => Mammal(x)).
  wff3!:  all(x)(BearsYoungAlive(x) => Mammal(x))

: LivesInWater(whale).
  wff4!:  LivesInWater(whale)

: BearsYoungAlive(whale).
  wff5!:  BearsYoungAlive(whale)
```

Contradictions

The Contradiction

```
: ?x(whale)?

  A contradiction was detected within context default-defaultct.
  The contradiction involves the newly derived proposition:
      wff6!:  Mammal(whale)

  and the previously existing proposition:
      wff7!:  ~Mammal(whale)

  You have the following options:
   1. [C]ontinue anyway, knowing that a contradiction is derivable;
   2. [R]e-start the exact same run in a different context which is
      not inconsistent;
   3. [D]rop the run altogether.

  (please type c, r or d)
=><= d
```

SNePSLOG Demonstrations

```
: demo
Available demonstrations:
  1:  Socrates - Is he mortal?
  2:  UVBR - Demonstrating the Unique Variable Binding Rule
  3:  The Jobs Puzzle - A solution with the Numerical Quantifie
  4:  Pegasus - Why winged horses lead to contradictions
  5:  Schubert's Steamroller
  6:  Rule Introduction - Various examples
  7:  Examples of various SNeRE constructs.
  8:  Enter a demo filename
Your choice (q to quit):
```

8.5 Reasoning Heuristics

Logically equivalent SNePSLOG wffs

are interpreted differently by the SNePS Reasoning System.

`v=>`-Elimination

Instead of

$$\frac{\texttt{P()} \qquad \texttt{(P() or Q()) => R()}}{\texttt{R()}}$$

which would require `or`-I followed by `=>`-E

Have

$$\frac{\texttt{P()} \qquad \texttt{\{P(), Q()\} v=> R()}}{\texttt{R()}}$$

which requires only `v=>`-E

Example of v=>-E

```
: P().
  wff1!:  P()

: {P(), Q()} v=> R().
  wff4!:  {Q(),P()} v=> {R()}

: trace inference
Tracing inference.

: R()?
I wonder if  wff3:   R()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   Q()
holds within the BS defined by context default-defaultct

I know  wff1!:  P()

Since  wff4!:  {Q(),P()} v=> {R()}
and  wff1!:  P()
I infer  wff3:   R()

  wff3!:  R()
```

Bi-Directional Inference
Backward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: p().
  wff1!:  p()

: r()?

I wonder if  wff3:   r()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   q()
holds within the BS defined by context default-defaultct

I know  wff1!:  p()

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff3!:  r()
```

Bi-Directional Inference
Forward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: p()!

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff4:   s()

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff4!:  s()
  wff3!:  r()
  wff1!:  p()
```

Bi-Directional Inference
Forward-in-Backward Inference

```
: {p(), q()} v=> {r(), s()}.
  wff5!:  {q(),p()} v=> {s(),r()}

: r()?

I wonder if  wff3:   r()
holds within the BS defined by context default-defaultct

I wonder if  wff2:   q()
holds within the BS defined by context default-defaultct

I wonder if  wff1:   p()
holds within the BS defined by context default-defaultct

: p()!

Since  wff5!:  {q(),p()} v=> {s(),r()}
and  wff1!:  p()
I infer  wff3:   r()

  wff3!:  r()
  wff1!:  p()
```

Active connection graph cleared by `clear-infer`.

Bi-Directional Inference
Backward-in-Forward Inference

```
: p().
  wff1!:  p()

: p() => (q() => r()).
  wff5!:  p() => (q() => r())

: q()!

I know  wff1!:  p()

Since  wff5!:  p() => (q() => r())
and  wff1!:  p()
I infer  wff4:   q() => r()

I know  wff2!:  q()

Since  wff4!:  q() => r()
and  wff2!:  q()
I infer  wff3:   r()

  wff4!:  q() => r()
  wff3!:  r()
  wff2!:  q()
```

Modus Tollens Not Implemented

```
: all(x)(p(x) => q(x)).
  wff1!:  all(x)(p(x) => q(x))

: p(a).
  wff2!:  p(a)

: q(a)?
  wff3!:  q(a)

: ~q(b).
  wff6!:  ~q(b)

: p(b)?
:
```

Use Disjunctive Syllogism Instead

```
: all(x)(or{~p(x), q(x)}).
  wff1!:  all(x)(q(x) or ~p(x))

: p(a).
  wff2!:  p(a)

: q(a)?
  wff3!:  q(a)

: ~q(b).
  wff7!:  ~q(b)

: p(b)?
  wff9!:  ~p(b)
```

=> Is Not Material Implication

If ⇒ is material implication,

$$\neg(P \Rightarrow Q) \Leftrightarrow (P \wedge \neg Q)$$

and

$$\neg(P \Rightarrow Q) \models P$$

But `~(p => q)` just means that its not the case that `p => q`:

```
: ~(p() => q()).
  wff4!:  ~(p() => q())

: p()?


:
```

Use `or` Instead of Material Implication

```
: ~(~p() or q()).
  wff5!:  nor{q(),~p()}

: p()?

  wff1!:  p()
```

Ordering of Nested Rules Matters
Optimal Order

```
: wifeOf(Caren,Stu).
: wifeOf(Ruth,Mike).
: brotherOf(Stu,Judi).
: brotherOf(Mike,Lou).
: parentOf(Judi,Ken).
: parentOf(Lou,Stu).
: all(w,x)(wifeOf(w,x)
          => all(y)(brotherOf(x,y)
                    => all(z)(parentOf(y,z)
                              => auntOf(w,z)))).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p7:   wifeOf(Caren,x)
I know  wff1!:  wifeOf(Caren,Stu)

I wonder if  p8:   brotherOf(Stu,y)
I know  wff3!:  brotherOf(Stu,Judi)

I wonder if  wff5!:  parentOf(Judi,Ken)
I know  wff5!:  parentOf(Judi,Ken)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.03
```

Ordering of Nested Rules Matters
Bad Order

```
all(x,y)(brotherOf(x,y)
         => all(w)(wifeOf(w,x)
                   => all(z)(parentOf(y,z)
                             => auntOf(w,z)))).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p1:   brotherOf(x,y)
I know  wff3!:  brotherOf(Stu,Judi)
I know  wff4!:  brotherOf(Mike,Lou)

I wonder if  wff12:   wifeOf(Caren,Mike)
I wonder if  wff1!:  wifeOf(Caren,Stu)
I know  wff1!:  wifeOf(Caren,Stu)

I wonder if  wff5!:  parentOf(Judi,Ken)
I know  wff5!:  parentOf(Judi,Ken)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.04
```

Ordering of Nested Rules Matters
Parallel

```
all(w,x,y,z)({wifeOf(w,x),brotherOf(x,y),parentOf(y,z)}
              &=> auntOf(w,z)).
: auntOf(Caren,Ken)?

I wonder if  wff8:   auntOf(Caren,Ken)
I wonder if  p5:   parentOf(y,Ken)
I wonder if  p2:   brotherOf(x,y)
I wonder if  p6:   wifeOf(Caren,x)

I know  wff5!:  parentOf(Judi,Ken)
I know  wff3!:  brotherOf(Stu,Judi)
I know  wff4!:  brotherOf(Mike,Lou)
I know  wff1!:  wifeOf(Caren,Stu)

  wff8!:  auntOf(Caren,Ken)
 CPU time : 0.03
```

Lemmas (Expertise)
Knowledge Base

```
: all(r)(transitive(r)
        => all(x,y,z)({r(x,y),r(y,z)} &=> r(x,z))).
: transitive(biggerThan).
: biggerThan(elephant,lion).
: biggerThan(lion,hyena).
: biggerThan(hyena,rat).
```

Lemmas: First Task

```
: biggerThan(?x,rat)?
I wonder if  p6:   biggerThan(x,rat)
I know  wff5!:  biggerThan(hyena,rat)
I wonder if  wff2!:  transitive(biggerThan)
I know  wff2!:  transitive(biggerThan)
I infer  wff6:   all(z,y,x)({biggerThan(x,y),biggerThan(y,z)} &=> {biggerThan(x,z)})
I wonder if  p8:   biggerThan(y,rat)
I wonder if  p10:   biggerThan(x,y)
I know  wff5!:  biggerThan(hyena,rat)
I wonder if  p12:   biggerThan(rat,z)
I know  wff3!:  biggerThan(elephant,lion)
I know  wff4!:  biggerThan(lion,hyena)
I infer  wff7:   biggerThan(lion,rat)
I infer  wff8:   biggerThan(elephant,rat)
...
  wff8!:  biggerThan(elephant,rat)
  wff7!:  biggerThan(lion,rat)
  wff5!:  biggerThan(hyena,rat)
 CPU time : 0.09
```

Second Task

```
: clear-infer
: biggerThan(truck,SUV).
: biggerThan(SUV,sedan).
: biggerThan(sedan,roadster).

: biggerThan(?x,roadster)?
I wonder if  p14:   biggerThan(x,roadster)
I know  wff11!:  biggerThan(sedan,roadster)
I wonder if  p10:   biggerThan(x,y)
I wonder if  p16:   biggerThan(y,roadster)
I know  wff3!:  biggerThan(elephant,lion)
I know  wff4!:  biggerThan(lion,hyena)
I know  wff5!:  biggerThan(hyena,rat)
I know  wff7!:  biggerThan(lion,rat)
I know  wff8!:  biggerThan(elephant,rat)
I know  wff9!:  biggerThan(truck,SUV)
I know  wff10!:  biggerThan(SUV,sedan)
I know  wff11!:  biggerThan(sedan,roadster)
I infer  wff12:   biggerThan(SUV,roadster)
I infer  wff13:   biggerThan(truck,roadster)
I wonder if  p17:   biggerThan(roadster,z)
  wff13!:  biggerThan(truck,roadster)
  wff12!:  biggerThan(SUV,roadster)
  wff11!:  biggerThan(sedan,roadster)
 CPU time : 0.04
```

Contexts

```
: demo /projects/shapiro/CSE563/Examples/SNePSLOG/facultyMeeting.snepslog
...
: ;;; Example of Contexts
;;; from
;;;   J. P. Martins & S. C. Shapiro, Reasoning in Multiple Belief Spaces IJCAI-83, 370-373.

: all(x)(meeting(x) => xor{time(x,morning), time(x,afternoon)}).
  wff1!:  all(x)(meeting(x) => (xor{time(x,afternoon),time(x,morning)}))
: all(x,y)({meeting(x),meeting(y)} &=> all(t)(xor{time(x,t),time(y,t)})).
  wff2!:  all(y,x)({meeting(y),meeting(x)} &=> {all(t)(xor{time(y,t),time(x,t)})})
: meeting(facultyMeeting).
  wff3!:  meeting(facultyMeeting)
: meeting(seminar).
  wff4!:  meeting(seminar)
: meeting(tennisGame).
  wff5!:  meeting(tennisGame)
: time(seminar,morning).
  wff6!:  time(seminar,morning)
: time(tennisGame,afternoon).
  wff7!:  time(tennisGame,afternoon)
: set-context stuSchedule {wff1,wff2,wff3,wff4,wff6}
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule)) (kinconsistent nil))
: set-context tonySchedule {wff1,wff2,wff3,wff5,wff7}
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule)) (kinconsistent nil))
: set-context patSchedule {wff1,wff2,wff3,wff4,wff5,wff6,wff7}
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1)) (named (patSchedule default-defaultct)) (kinconsistent nil))
```

Stu's Schedule

```
: set-default-context stuSchedule
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule))
 (kinconsistent nil))

: list-asserted-wffs
  wff6!:  time(seminar,morning)
  wff4!:  meeting(seminar)
  wff3!:  meeting(facultyMeeting)
  wff2!:  all(y,x)({meeting(y),meeting(x)}
                   &=> {all(t)(xor{time(y,t),time(x,t)})})
  wff1!:  all(x)(meeting(x)
                 => (xor{time(x,afternoon),time(x,morning)}))

: time(facultyMeeting,?t)?
  wff10!:  time(facultyMeeting,afternoon)
  wff9!:  ~time(facultyMeeting,morning)
```

Tony's Schedule

```
: set-default-context tonySchedule
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule))
 (kinconsistent nil))

: list-asserted-wffs
  wff12!:  xor{time(facultyMeeting,afternoon),time(facultyMeeting,morning)}
  wff7!:  time(tennisGame,afternoon)
  wff5!:  meeting(tennisGame)
  wff3!:  meeting(facultyMeeting)
  wff2!:  all(y,x)({meeting(y),meeting(x)}
                 &=> {all(t)(xor{time(y,t),time(x,t)})})
  wff1!:  all(x)(meeting(x)
                => (xor{time(x,afternoon),time(x,morning)}))

: time(facultyMeeting,?t)?
  wff11!:  ~time(facultyMeeting,afternoon)
  wff8!:  time(facultyMeeting,morning)
```

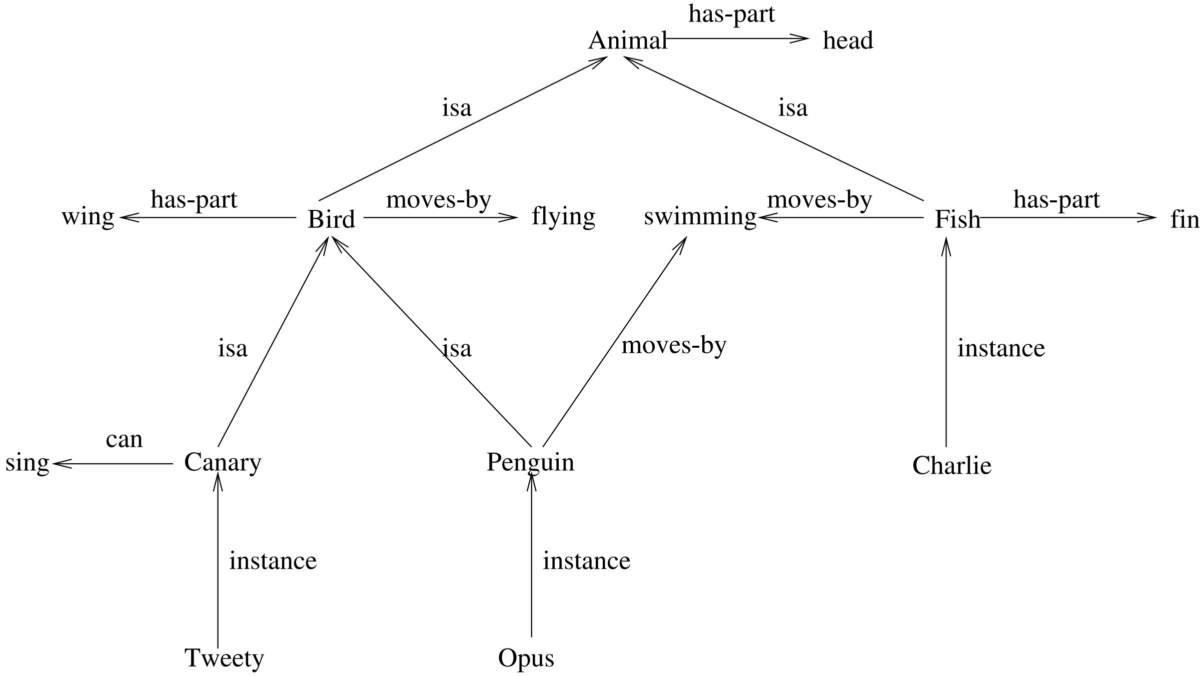
Pat’s Schedule

```
: set-default-context patSchedule
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (patSchedule default-defaultct)) (kinconsistent nil))

: time(facultyMeeting,?t)?

  A contradiction was detected within context patSchedule.
  The contradiction involves the newly derived proposition:
      wff8!:  time(facultyMeeting,morning)

  and the previously existing proposition:
      wff9!:  ~time(facultyMeeting,morning)

  You have the following options:
   1. [C]ontinue anyway, knowing that a contradiction is derivable;
   2. [R]e-start the exact same run in a different context which is
      not inconsistent;
   3. [D]rop the run altogether.

  (please type c, r or d)
=><= d
```

Resulting Contexts

```
: describe-context stuSchedule
((assertions (wff6 wff4 wff3 wff2 wff1)) (named (stuSchedule))
 (kinconsistent nil))

: describe-context tonySchedule
((assertions (wff7 wff5 wff3 wff2 wff1)) (named (tonySchedule))
 (kinconsistent nil))

: describe-context patSchedule
((assertions (wff7 wff6 wff5 wff4 wff3 wff2 wff1))
 (named (patSchedule default-defaultct)) (kinconsistent t))
```

8.6 SNePS as a Network: Semantic Networks



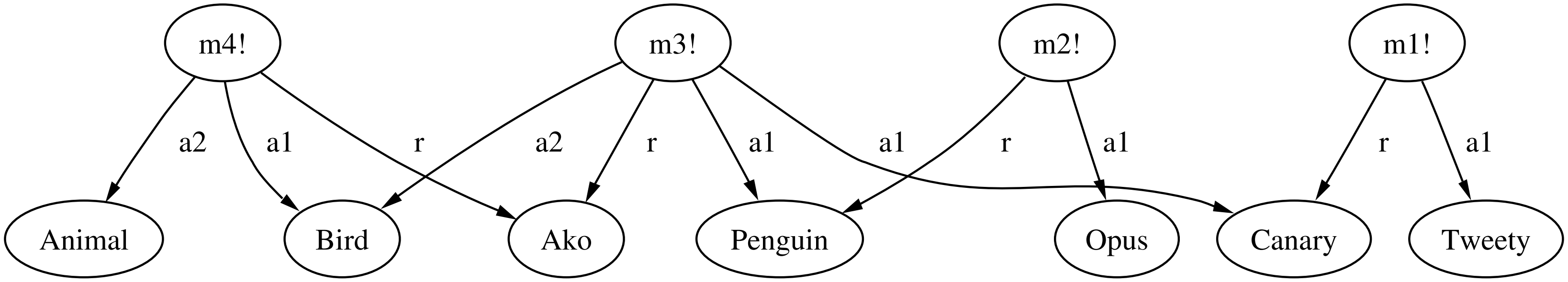
Some psychological evidence.

More efficient search than logical inference.

Unclear semantics.

SNePS as a Network

```
: clearkb
: Canary(Tweety).
: Penguin(Opus).
: Ako({Canary,Penguin}, Bird).
: Ako(Bird, Animal).
: show
```



Defining Case Frames

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...

: define-frame Canary(class member) "[member] is a [class]"
Canary(x1) will be represented by {<class, Canary>, <member, x1>}

: define-frame Penguin(class member) "[member] is a [class]"
Penguin(x1) will be represented by {<class, Penguin>, <member, x1>}

: define-frame Ako(nil subclass superclass) "Every [subclass] is a [superclas
Ako(x1, x2) will be represented by {<subclass, x1>, <superclass, x2>}
```

Entering the KB

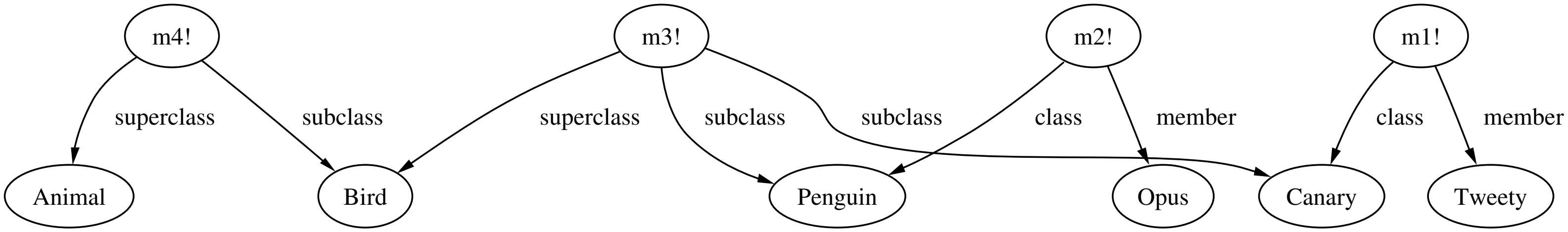
```
: Canary(Tweety).
  wff1!:  Canary(Tweety)

: Penguin(Opus).
  wff2!:  Penguin(Opus)

: Ako({Canary, Penguin}, Bird).
  wff3!:  Ako({Penguin,Canary},Bird)

: Ako(Bird, Animal).
  wff4!:  Ako(Bird,Animal)
```

The Knowledge Base

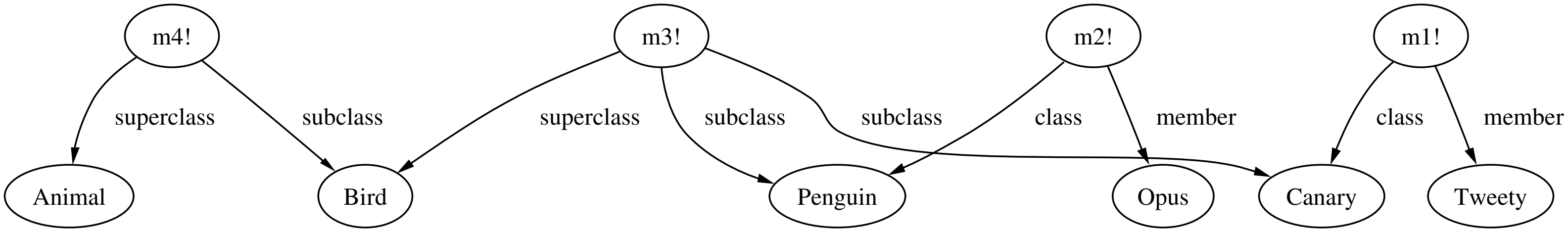
```
: list-terms
  wff1!:  Canary(Tweety)
  wff2!:  Penguin(Opus)
  wff3!:  Ako({Penguin,Canary},Bird)
  wff4!:  Ako(Bird,Animal)

: describe-terms
Tweety is a Canary.
Opus is a Penguin.
Every Penguin and Canary is a Bird.
Every Bird is a Animal.
```

The Network

`: show`



Path-Based Inference



```
: define-path class (compose class
                       (kstar (compose subclass- ! superclass))
class implied by the path (compose class
                           (kstar
                            (compose subclass- ! superclass)))
class- implied by the path (compose
                            (kstar (compose superclass-
                                            ! subclass))
                            class-)
```

Using Path-Based Inference

```
: list-asserted-wffs
  wff4!:  Ako(Bird,Animal)
  wff3!:  Ako({Penguin,Canary},Bird)
  wff2!:  Penguin(Opus)
  wff1!:  Canary(Tweety)

: define-frame Animal(class member) "[member] is a [class]"
Animal(x1) will be represented by {<class, Animal>, <member, x1>}

: trace inference
Tracing inference.

: Animal(Tweety)?
I wonder if  wff5:   Animal(Tweety)
holds within the BS defined by context default-defaultct
I know  wff1!:  Canary(Tweety)
  wff5!:  Animal(Tweety)
```

Rules About Functions in Mode 3

```
: set-mode-3
: define-frame WestOf(relation domain range)
: define-frame isAbove(relation domain range)
: define-frame Likes(relation liker likee)

: define-frame r(relation domain range)
: define-frame anti-symmetric(nil antisymm)

: all(r)(anti-symmetric(r) => all(x,y)(r(x,y) => ~r(y,x))).
  wff1!:  all(r)(anti-symmetric(r) => (all(y,x)(r(x,y) => (~r(y,x)))))

: anti-symmetric({WestOf, isAbove, Likes}).

: WestOf(Buffalo,Rochester).
: isAbove(penthouse37,lobby37).
: Likes(Betty,Tom).

: WestOf(?x,?y)?
  wff9!:  ~WestOf(Rochester,Buffalo)
  wff3!:  WestOf(Buffalo,Rochester)

: isAbove(?x,?y)?
  wff13!:  ~isAbove(lobby37,penthouse37)
  wff4!:  isAbove(penthouse37,lobby37)

: Likes(?x,?y)?
  wff5!:  Likes(Betty,Tom)
```

Procedural Attachment in SNePS

```
cl-user(3): (snepslog)
: load /projects/snwiz/Libraries/expressions.snepslog

: define-frame Value(nil obj val) "the value of [obj] is [val]"

: define-frame radius(nil radiusof) "the radius of [radiusof]"

: define-frame volume(nil volumeof) "the volume of [volumeof]"

: all(x,r,p)({Value(radius(x), r), Value(pi,p)}
           &=> all(v)(is(v,/(*(4.0,*(p,*(r,*(r,r)))),3.0))
                      => Value(volume(x),v))).

: Value(pi,3.14159).

: Value(radius(sphere1), 9.0).

: Value(volume(sphere1), ?x)?
  wff13!:  Value(volume(sphere1),3053.6257)
```

8.7 SNeRE: The SNePS Rational Engine

Motivation

Coming to believe something
is different from acting.

Prolog Searches In Order
The KB

```
| ?- [user].
% consulting user...
| q(X) :- q1(X), q2(X).
| q1(X) :- p(X), s(X).
| q2(X) :- r(X), s(X).
| s(X) :- t(X).
| p(a).
| r(a).
| t(a).
|
% consulted user in module user, 0 msec 1592 bytes
yes
```

Prolog Searches In Order

The Run

```
| ?- trace.
% The debugger will first creep -- showing everything (trace)
yes
% trace
| ?- q(a).
        1      1 Call: q(a) ?
        2      2 Call: q1(a) ?
        3      3 Call: p(a) ?
        3      3 Exit: p(a) ?
        4      3 Call: s(a) ?
        5      4 Call: t(a) ?
        5      4 Exit: t(a) ?
        4      3 Exit: s(a) ?
        2      2 Exit: q1(a) ?
        6      2 Call: q2(a) ?
        7      3 Call: r(a) ?
        7      3 Exit: r(a) ?
        8      3 Call: s(a) ?
        9      4 Call: t(a) ?
        9      4 Exit: t(a) ?
        8      3 Exit: s(a) ?
        6      2 Exit: q2(a) ?
        1      1 Exit: q(a) ?
yes
```

SNePS Avoids Extra Search

The KB

```
: clearkb
Knowledge Base Cleared

: all(x)({q1(x), q2(x)} &=> q(x)).
: all(x)({p(x), s(x)} &=> q1(x)).
: all(x)({r(x), s(x)} &=> q2(x)).
: all(x)(t(x) => s(x)).

: p(a).
: r(a).
: t(a).
```

SNePS Avoids Extra Search

The Search

```
: trace inference
Tracing inference.

: q(a)?
I wonder if  wff8:   q(a)
I wonder if  wff10:   q2(a)
I wonder if  wff12:   q1(a)
I wonder if  wff14:   s(a)
I wonder if  wff6!:  r(a)
I wonder if  wff14:   s(a)
I wonder if  wff5!:  p(a)
I know  wff6!:  r(a)
I know  wff5!:  p(a)
I wonder if  wff7!:  t(a)
I know  wff7!:  t(a)
```

SNePS Avoids Extra Search
The Answers

```
Since  wff4!:  all(x)(t(x) => s(x))
and  wff7!:  t(a)
I infer  wff14:   s(a)

Since  wff3!:  all(x)({s(x),r(x)} &=> {q2(x)})
and  wff14!:  s(a)
and  wff6!:  r(a)
I infer  wff10:   q2(a)

Since  wff2!:  all(x)({s(x),p(x)} &=> {q1(x)})
and  wff14!:  s(a)
and  wff5!:  p(a)
I infer  wff12:   q1(a)

Since  wff1!:  all(x)({q2(x),q1(x)} &=> {q(x)})
and  wff10!:  q2(a)
and  wff12!:  q1(a)
I infer  wff8:   q(a)

  wff8!:  q(a)
```

Primitive Acts

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...

: define-frame say(action line)
say(x1) will be represented by {<action, say>, <line, x1>}

: ^^
--> (define-primaction sayaction ((line))
      (format sneps:outunit "~A" line))
sayaction

--> (attach-primaction say sayaction)
t

--> ^^

: perform say("Hello world")
Hello world
```

Effects: The KB

```
: set-mode-3
Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.
...
Effect(x1, x2) will be represented by {<act, x1>, <effect, x2>}
...

: define-frame say (action line)
: define-frame said (act agent object)
: define-frame Utterance (class member)
: ^^
--> (define-primaction sayaction ((line))
      (format sneps:outunit "~A" line))
sayaction
-->
(attach-primaction say sayaction)
t
--> ^^
: Utterance("Hello world").
: all(x)(Utterance(x) => Effect(say(x), said(I,x))).
```

Effects: The Run

```
: list-asserted-wffs
  wff2!:  all(x)(Utterance(x) => Effect(say(x),said(I,x)))
  wff1!:  Utterance(Hello world)

: perform say("Hello world")
Hello world

: list-asserted-wffs
  wff5!:  Effect(say(Hello world),said(I,Hello world))
  wff4!:  said(I,Hello world)
  wff2!:  all(x)(Utterance(x) => Effect(say(x),said(I,x)))
  wff1!:  Utterance(Hello world)
```

Defined Acts

```
: set-mode-3
...
ActPlan(x1, x2) will be represented by {<act, x1>, <plan, x2>}
...

: define-frame say (action part1 part2)
: define-frame greet (action object)
: define-frame Person (class member)

: ^^
--> (define-primaction sayaction ((part1) (part2))
      (format sneps:outunit "~A ~A~%"
              part1 part2))
sayaction
-->
(attach-primaction say sayaction)
t
--> ^^

: all(x)(Person(x) => ActPlan(greet(x), say(Hello,x))).
: Person(Mike).

: perform greet(Mike).
Hello Mike
```

Other Propositions about Acts

`GoalPlan(`p`, `a`)`

`Precondition(`a`, `p`)`

Control Acts

`achieve(`*p*`)`

`do-all({`*a1, ..., an*`})`

`do-one({`*a1, ..., an*`})`

`snif({if(`*p1,a1*`), ..., if(`*pn,an*`)[, else(`*da*`)]})`

`sniterate({if(`*p1,a1*`), ..., if(`*pn,an*`)[, else(`*da*`)]})`

`snsequence(`*a1, a2*`)`

`withall(`*x, p(x), a(x)*`[, `*da*`])`

`withsome(`*x, p(x), a(x)*`[, `*da*`])`

Must use `attach-primaction` on whichever you want to use.

Policies

`ifdo(`p`, `a`)`

`whendo(`p`, `a`)`

`wheneverdo(`p`, `a`)`

Mental Acts

`believe(`p`)`

`disbelieve(`p`)`

`adopt(`p`)`

`unadopt(`p`)`

The Execution Cycle

```
perform(act):
   pre := $\{p$ | Precondition(act,$p$)$\}$;
   notyet := pre - $\{p$ | $p \in$ pre & $\vdash p\}$;
   if notyet $\neq$ nil
      then perform(snsequence(do-all($\{a$ | $p \in$ notyet
                                                     & $a$ = achieve($p$)$\}$),
                                  act))
      else $\{$effects := $\{p$ | Effect(act,$p$)$\}$;
            if act $\textit{is primitive}$
               then apply(primitive-function(act), objects(act));
               else perform(do-one($\{p$ | ActPlan(act,$p$)$\}$))
            believe(effects)$\}$
```

Examples

`SNePSLOG` demo #7

`/projects/robot/Karel/ElevatorWorld/elevator.snepslog`

`/projects/robot/Karel/DeliveryWorld/DeliveryAgent.snepslog`

`/projects/robot/Karel/WumpusWorld/WWAgent.snepslog`

`/projects/robot/Fevahr/Ascii/afevahr.snepslog`

`/projects/robot/Fevahr/Java/jfevahr.snepslog`

`/projects/robot/Greenfoot/ElevatorWorld/sneps/elevator.snepslog`

`/projects/robot/Greenfoot/WumpusWorld/sneps/WWAgent.snepslog`

9 Belief Revision/Truth Maintenance

9.1 Motivation

Floors Above and Below Ground

```
: xor{OnFloor(1),OnFloor(2),OnFloor(3),OnFloor(4)}.
: {OnFloor(1), OnFloor(2)} => {Location(belowGround)}.
: {OnFloor(3), OnFloor(4)} => {Location(aboveGround)}.

: perform believe(OnFloor(1))

: list-asserted-wffs
  wff13!:  ~OnFloor(2)
  wff12!:  ~OnFloor(3)
  wff11!:  ~OnFloor(4)
  wff9!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
  wff7!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
  wff6!:  Location(belowGround)
  wff5!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
  wff1!:  OnFloor(1)
```

Motivation
Disbelieving an Hypothesis

```
: perform disbelieve(OnFloor(1))

: list-asserted-wffs
  wff9!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
  wff7!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
  wff5!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
```

Note the absence of `Location(belowGround)`

Moral

If retain derived beliefs (lemmas),

need a way to delete them
when their foundations are removed.

When Needed 1

If the KB contains beliefs about the (some) world,

and the world changes,

and the KB does not have a model of time.

I.e. the beliefs in the KB are of the form,
I believe this is true now.

What's needed

Links from hypotheses to propositions derived from them.

`=>` *vs.* `when(ever)do`: Assertions

```
: Floor({1,2,3,4}).
: xor{OnFloor(1),OnFloor(2),OnFloor(3),OnFloor(4)}.
: {OnFloor(1), OnFloor(2)} => {Location(belowGround)}.
: {OnFloor(3), OnFloor(4)} => {Location(aboveGround)}.
: perform withall(f, Floor(f),
                adopt(wheneverdo(OnFloor(f),
                      believe(HaveBeenOnFloor(f)))),
                noop()).
: perform believe(OnFloor(1))
```

`=>` ***vs.*** `when(ever)do`: The KB

```
: list-asserted-wffs
  wff37!:  ~OnFloor(2)
  wff36!:  ~OnFloor(3)
  wff35!:  ~OnFloor(4)
  wff31!:  wheneverdo(OnFloor(4),believe(HaveBeenOnFloor(4)))
  wff27!:  wheneverdo(OnFloor(3),believe(HaveBeenOnFloor(3)))
  wff23!:  wheneverdo(OnFloor(2),believe(HaveBeenOnFloor(2)))
  wff19!:  wheneverdo(OnFloor(1),believe(HaveBeenOnFloor(1)))
  wff17!:  HaveBeenOnFloor(1)
  wff16!:  Floor(1)
  wff15!:  Floor(2)
  wff14!:  Floor(3)
  wff13!:  Floor(4)
  wff10!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
  wff8!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
  wff7!:  Location(belowGround)
  wff6!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
  wff2!:  OnFloor(1)
  wff1!:  Floor({4,3,2,1})
```

`=>` *vs.* `when(ever)do`: Move Floors

```
: perform believe(OnFloor(4))

: list-asserted-wffs
  wff39!:  ~OnFloor(1)
  wff37!:  ~OnFloor(2)
  wff36!:  ~OnFloor(3)
  wff31!:  wheneverdo(OnFloor(4),believe(HaveBeenOnFloor(4)))
  wff29!:  HaveBeenOnFloor(4)
  wff27!:  wheneverdo(OnFloor(3),believe(HaveBeenOnFloor(3)))
  wff23!:  wheneverdo(OnFloor(2),believe(HaveBeenOnFloor(2)))
  wff19!:  wheneverdo(OnFloor(1),believe(HaveBeenOnFloor(1)))
  wff17!:  HaveBeenOnFloor(1)
  wff16!:  Floor(1)
  wff15!:  Floor(2)
  wff14!:  Floor(3)
  wff13!:  Floor(4)
  wff10!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
  wff9!:  Location(aboveGround)
  wff8!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
  wff6!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
  wff5!:  OnFloor(4)
  wff1!:  Floor({4,3,2,1})
```

`HaveBeenOnFloor(1)` remains; `OnFloor(1)` doesn’t.

Moral

The consequents of
`=>`, `v=>`, `&=>`, `or`, `nand`, `xor`, `iff`, `andor`, `thresh`, and `nexists`
are derived and retain a connection to their underlying hypotheses.

Whatever is `believe`'d is a hypothesis.

Use `=>`, `v=>`, `&=>`, `or`, `nand`, `xor`, `iff`, `andor`, `thresh`, and `nexists` for logical implications.

Use `whendo(`*`p1,believe(p2)`*`)` or `wheneverdo(`*`p1,believe(p2)`*`)` for decisions.

Contingent Plans

```
: xor{Location(BellHall), Location(home)}.
: Location(BellHall) => ActPlan(getMail, go(MailRoom)).
: Location(home) => ActPlan(getMail, go(mailBox)).

: perform believe(Location(BellHall))
: ActPlan(getMail, ?how)?
  wff5!:  ActPlan(getMail,go(MailRoom))

: perform believe(Location(home))
: ActPlan(getMail, ?how)?
  wff8!:  ActPlan(getMail,go(mailBox))
```

Moral

Using this design for contingent plans,
along with retention of lemmas,
depends on belief revision.

Motivation

Sea Creatures

```
: all(x)(andor(0,1){Ako(x, mammal), Ako(x, fish)}).

: all(x)(LiveIn(x, water) => Ako(x, fish)).

: all(x)(BearYoung(x, live) => Ako(x, mammal)).

: LiveIn(whales, water).
: LiveIn(sharks, water).

: BearYoung(whales, live).
: BearYoung(dogs, live).
```

Motivation

Are Whales Fish or Mammals?

```
: Ako(whales, ?x)?

A contradiction was detected within context default-defaultct
The contradiction involves the newly derived proposition:
    wff8!:  Ako(whales,mammal)

and the previously existing proposition:
    wff9!:  ~Ako(whales,mammal)
```

SNeBR Options

```
You have the following options:
 1. [C]ontinue anyway, knowing that a contradiction is derivabl
 2. [R]e-start the exact same run in a different context which
    not inconsistent;
 3. [D]rop the run altogether.

  (please type c, r or d)
=><= r

In order to make the context consistent you must delete at leas
one hypothesis from each of the following sets of hypotheses:
  (wff6 wff4 wff3 wff2 wff1)
```

Possible Culprits

```
 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.
An inconsistent set of hypotheses:
 1 :   wff6!:  BearYoung(whales,live)
          (2 supported propositions: (wff8 wff6) )
 2 :   wff4!:  LiveIn(whales,water)
          (3 supported propositions: (wff10 wff9 wff4) )
 3 :   wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal))
          (2 supported propositions: (wff8 wff3) )
 4 :   wff2!:  all(x)(LiveIn(x,water) => Ako(x,fish))
          (3 supported propositions: (wff10 wff9 wff2) )
 5 :   wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
          (2 supported propositions: (wff9 wff1) )
```

Choosing the Culprit

```
  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 4
```

Remaining Possible Culprits

```
The consistent set of hypotheses:
 1 :   wff6!:  BearYoung(whales,live)
          (2 supported propositions: (wff8 wff6) )

 2 :   wff4!:  LiveIn(whales,water)
          (1 supported proposition: (wff4) )

 3 :   wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal))
          (2 supported propositions: (wff8 wff3) )

 4 :   wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
          (1 supported proposition: (wff1) )
  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions
  (please type a number OR d, a, r, q or i)
=><= q
```

Other Hypotheses

```
  The following (not known to be inconsistent) set of
  hypotheses was also part of the context where the
  contradiction was derived:
    (wff7 wff5)
  Do you want to inspect or discard some of them?
=><= no
  Do you want to add a new hypothesis?  no

  wff11!:  ~Ako(whales,fish)
  wff8!:  Ako(whales,mammal)

 CPU time : 0.03
```

Resultant KB

```
: list-asserted-wffs
  wff12!:  ~(all(x)(LiveIn(x,water) => Ako(x,fish)))
  wff11!:  ~Ako(whales,fish)
  wff8!:  Ako(whales,mammal)
  wff7!:  BearYoung(dogs,live)
  wff6!:  BearYoung(whales,live)
  wff5!:  LiveIn(shakes,water)
  wff4!:  LiveIn(whales,water)
  wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal))
  wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
```

Moral
When Needed 2

If accepting information from multiple sources,

or just one possibly inconsistent source,

need a way to recognize contradictions,

and to find the culprit,

and to delete it,

and its implications.

What's Needed

Links between derived propositions

and hypotheses they were derived from.

9.2 Relevance Logic (R)

Motivation

Paradoxes of Implication 1

Anything Implies a Truth

1	$\vert\ A$	Hyp
2	$\vert\ \vert\ B$	Hyp
3	$\vert\ \vert\ A$	Reit, 1
4	$\vert\ B \Rightarrow A$	$\Rightarrow$ I, 2–3
5	$A \Rightarrow (B \Rightarrow A)$	$\Rightarrow$ I, 1–4

But it seems that B had nothing to do with deriving A.

Motivation of R

Paradoxes of Implication 2

A Contradiction Implies Anything

1	$\mid A \wedge \neg A$	Hyp
2	$\mid \; \mid \neg B$	Hyp
3	$\mid \; \mid A \wedge \neg A$	Reit, 1
4	$\mid \; \mid A$	$\wedge$E, 3
5	$\mid \; \mid \neg A$	$\wedge$E, 3
6	$\mid B$	$\neg$I, 2–5
7	$(A \wedge \neg A) \Rightarrow B$	$\Rightarrow$I, 1–6

But it seems that $\neg B$ had nothing to do with deriving the contradiction.

What's Needed

A way to determine when a hypothesis is really used to derive another wff.

When a hypothesis is **relevant** to a conclusion.

9.3 R

Relevance Logic

The Logic of Relevant Implication

Syntax: The same as Standard FOL.

Intensional Semantics: The same as Standard FOL.

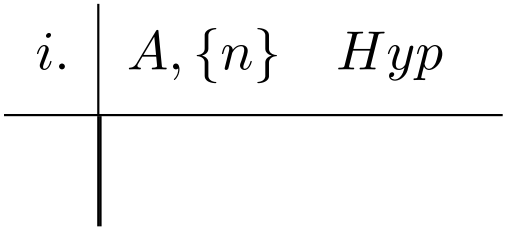
Extensional Semantics: The same as Standard FOL for terms.
For wffs: a four-valued logic, using True, False, Neither, and Both.

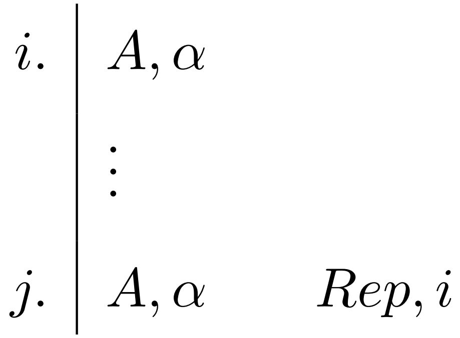
KB Interpretations of R's 4 Truth Values

True	true
False	false
Neither	unknown
Both	contradictory, "I've been told both." or a "true contradiction" such as Russell's set both is and isn't a member of itself.

9.4 R Proof Theory

Structural Rules of Inference





where n is a new integer.

R Rules for $\Rightarrow$

$$\begin{array}{r|l|ll}
i. & & A, \{n\} & Hyp \\ \cline{3-4}
 & & \vdots & \\
j. & & B, \alpha,\ s.t.\ n \in \alpha & \\
k & \multicolumn{2}{l}{(A \Rightarrow B), \alpha - \{n\}} & \Rightarrow I, i\text{–}j
\end{array}
\qquad\qquad
\begin{array}{r|ll}
i. & A, \alpha & \\
 & \vdots & \\
j. & (A \Rightarrow B), \beta & \\
k. & B, \alpha \cup \beta & \Rightarrow E, i, j
\end{array}$$

How the Paradoxes of Implication are Blocked 1

1. $A, \{1\}$ *Hyp*

2. $B, \{2\}$ *Hyp*

3. $A, \{1\}$ *Reit, 1*

Can't then apply $\Rightarrow I$

R Rules for $\wedge$

$$\begin{array}{r|ll} i_1. & A_1, \alpha & \\ & \vdots & \\ i_n. & A_n, \alpha & \\ j. & A_1 \wedge \cdots \wedge A_n, \alpha & \wedge I, i_1, \ldots, i_n \end{array}$$

$$\begin{array}{r|ll} i. & A_1 \wedge \cdots \wedge A_n, \alpha & \\ & \vdots & \\ j. & A_k, \alpha & \wedge E, i \end{array}$$

Why $\land I$ Requires the Same OS

If Not

1	$A, \{1\}$	Hyp, 2–5
2	$B, \{2\}$	Hyp, 3–5
3	$A, \{1\}$	Reit, 1
4	$(A \land B), \{1, 2\}$	∧I?
5	$A, \{1, 2\}$	∧E, 4
6	$(B \Rightarrow A), \{1\}$	⇒I, 2–5
7	$(A \Rightarrow (B \Rightarrow A)), \{\}$	⇒I, 1–6

Reconstruct paradox of implication.

Note: Empty os means a theorem.

Extended Rule for $\wedge I$

$$\begin{array}{r|ll}
i_1. & A_1, \alpha & \\
 & \vdots & \\
i_n. & A_n, \eta & \\
j. & A_1 \wedge \cdots \wedge A_n, (\alpha \cup \cdots \cup \eta)^* & \wedge I, i_1, \ldots, i_n
\end{array}$$

Can't apply $\wedge E$ to an extended wff.

R Rules for ¬

$$
\begin{array}{r|l|ll}
i. & & A, \{n\} & Hyp \\ \cline{3-4}
 & & \vdots & \\
j. & & B, \alpha \ s.t.\ n \in \alpha & \\
j+1. & & \neg B, \alpha & \\
j+2. & & \neg A, \alpha - \{n\} & \neg I, i\text{–}(j+1)
\end{array}
\qquad
\begin{array}{r|l|ll}
i. & & \neg A, \{n\} & Hyp \\ \cline{3-4}
 & & \vdots & \\
j. & & B, \alpha \ s.t.\ n \in \alpha & \\
j+1. & & \neg B, \alpha & \\
j+2. & & A, \alpha - \{n\} & \neg I, i\text{–}(j+1
\end{array}
$$

$$
\begin{array}{r|ll}
i. & \neg\neg A, \alpha & \\
j. & A, \alpha & \neg E, i
\end{array}
$$

Extended R Rule for $\neg I$

$i.$	$A, \{n\}$	Hyp
	$\vdots$	
$j.$	B, α	
$j+1.$	$\neg B, \beta$	
$j+2.$	$\neg A, ((\alpha \cup \beta) - \{n\})^*\ s.t.\ n \in (\alpha \cup \beta)$	$\neg I, i\text{–}(j+1)$

$i.$	$\neg A, \{n\}$	Hyp
	$\vdots$	
$j.$	B, α	
$j+1.$	$\neg B, \beta$	
$j+2.$	$A, ((\alpha \cup \beta) - \{n\})^*\ s.t.\ n \in (\alpha \cup \beta)$	$\neg I, i\text{–}(j+1)$

How the Paradoxes of Implication are Blocked 2

1.	$(A \wedge \neg A), \{1\}$	*Hyp*
2.	$\neg B, \{2\}$	*Hyp*
3.	$(A \wedge \neg A), \{1\}$	*Reit*, *1*
4.	$A, \{1\}$	$\wedge E$, *3*
5.	$\neg A, \{1\}$	$\wedge E$, *3*

Can't then apply $\neg I$

R is a **paraconsistent** logic:
a contradiction does not imply anything whatsoever.

R Rule for $\vee I$

$$\begin{array}{r|l} i. & A_i, \alpha \\ j. & A_1 \vee \cdots \vee A_i \vee \cdots \vee A_n, \alpha \quad \vee I, i \end{array}$$

R Rule for $\vee E$

$$\begin{array}{r|ll}
i_1. & A_1 \vee \cdots \vee A_n, \alpha & \\
 & \vdots & \\
i_2. & A_1 \Rightarrow B, \beta & \\
 & \vdots & \\
i_3. & A_n \Rightarrow B, \beta & \\
j. & B, \alpha \cup \beta & \qquad \vee E, i_1, i_2, i_3
\end{array}$$

Irrelevance of Disjunctive Syllogism

1	$((A \vee B) \wedge \neg A), \{1\}$	Hyp	
2	$\neg A, \{1\}$	$\wedge$E,1	
3	$(A \vee B), \{1\}$	$\wedge$E,1	
4	$A, \{2\}$	Hyp	
5	$\neg B, \{3\}$	Hyp	
6	$A, \{2\}$	Reit, 4	
7	$\neg A, \{1\}$	Reit, 2	
8	B	$\neg$I, 5–7	**Not valid in R**
9	$A \Rightarrow B$	$\Rightarrow$I, 4–8	
10	$B, \{4\}$	Hyp	
11	$B, \{4\}$	Rep, 10	
12	$B \Rightarrow B, \{\}$	$\Rightarrow$I, 10–11	
13	B	$\vee$E, 3,9,12	

So $\vee$ is just truth-functional.

R Rules for Intensional OR ($\oplus$)

$$\begin{array}{r|ll}
i. & (\neg A \Rightarrow B), \alpha & \\
 & \vdots & \\
j. & (\neg B \Rightarrow A), \alpha & \\
j+1. & (A \oplus B), \alpha & \oplus I, i, j
\end{array}$$

$$\begin{array}{r|ll}
i. & (A \oplus B), \alpha & \\
 & \vdots & \\
j. & \neg A, \beta & \\
j+1. & B, \alpha \cup \beta & \oplus E
\end{array}
\qquad\qquad
\begin{array}{r|ll}
i. & (A \oplus B), \alpha & \\
 & \vdots & \\
j. & \neg B, \beta & \\
j+1. & A, \alpha \cup \beta & \oplus E
\end{array}$$

R Rules for $\Leftrightarrow$

$$\begin{array}{r|ll} i. & (A \Rightarrow B), \alpha & \\ & \vdots & \\ j. & (B \Rightarrow A), \alpha & \\ j+1. & (A \Leftrightarrow B), \alpha & \Leftrightarrow I, i, j \end{array}$$

$$\begin{array}{r|ll} i. & A, \alpha & \\ & \vdots & \\ j. & (A \Leftrightarrow B), \beta & \\ j+1. & B, \alpha \cup \beta & \Leftrightarrow E, i, j \end{array} \qquad \begin{array}{r|ll} i. & B, \alpha & \\ & \vdots & \\ j. & (A \Leftrightarrow B), \beta & \\ j+1. & A, \alpha \cup \beta & \Leftrightarrow E, i, j \end{array}$$

R Rules for ∀

$$
\begin{array}{r|l|ll}
i. & & A(a), \{n\} & Hyp \\ \cline{3-4}
 & & \vdots & \\
j. & & B(a), \alpha\ s.t.\ n \in \alpha & \\
j+1. & \multicolumn{3}{|l}{\quad \forall x(A(x) \Rightarrow B(x)), \alpha - \{n\} \quad \forall I, i\text{–}j}
\end{array}
$$

$$
\begin{array}{r|ll}
i. & A(t), \alpha & \\
 & \vdots & \\
j. & \forall x(A(x) \Rightarrow B(x)), \beta & \\
j+1. & B(t), \alpha \cup \beta & \forall E, i, j
\end{array}
$$

Where a is an arbitrary individual not otherwise used in the proof, and t is free for x in $B(x)$.

Note ∀ only governs ⇒.

R Rules for ∃

$$\begin{array}{r|l}
i & A(t),\alpha \\
i+1 & \exists x A(x),\alpha \quad \exists I, i
\end{array}$$

$$\begin{array}{r|l|ll}
i & \multicolumn{1}{l}{} & \exists x A(x),\alpha & \\
 & \multicolumn{1}{l}{} & \vdots & \\
j & & A\{a/x\},\beta & \text{Indef I}, i \\ \cline{2-4}
 & & \vdots & \\
k & & B,\gamma\ s.t.\ \beta\subset\gamma & \\
k+1 & \multicolumn{1}{l}{} & B,\gamma-\beta & \exists E, j\text{–}k
\end{array}$$

Where $A(x)$ is the result of replacing some or all occurrences of t in $A(t)$ by x,

t is free for x in $A(x)$;

a is an indefinite individual not otherwise used in the proof,

$A(a/x)$ is the result of replacing all occurrences of x in $A(x)$ by a,

and there is no occurrence of a in B.

Why the Subproof Contours?

1. To keep track of assumptions for each derived wff.
   But this is accomplished by os.
2. To differentiate hypotheses from derived wffs.
   Introduce support: $\langle\{hyp \mid der \mid ext\}, os\rangle$
   with origin tag and origin set.

SNePS KB

The SNePS KB consists of a collection of supported wffs.

A wff may have more than one support if it was derived in multiple ways.

Every implemented rule of inference specifies how the derived wff is derived from its parent(s) and how its support is derived from the support(s) of its parent(s).

Contexts and Belief Spaces

A context is a set of hypotheses.

A belief space defined by a context c is the set containing every wff whose os is a subset of c.

SNePSLOG Example

```
: expert
...
: xor{OnFloor(1),OnFloor(2),OnFloor(3),OnFloor(4)}.
  wff5!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
          {<hyp,{wff5}>}

: {OnFloor(1), OnFloor(2)} => {Location(belowGround)}.
  wff7!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
          {<hyp,{wff7}>}

: {OnFloor(3), OnFloor(4)} => {Location(aboveGround)}.
  wff9!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
          {<hyp,{wff9}>}
```

```
: perform believe(OnFloor(1))

: describe-context
((assertions (wff9 wff7 wff5 wff1))
 (named (default-defaultct)) (kinconsistent nil))
```

```
: list-asserted-wffs
  wff13!:  ~OnFloor(2)  {<der,{wff1,wff5}>}
  wff12!:  ~OnFloor(3)  {<der,{wff1,wff5}>}
  wff11!:  ~OnFloor(4)  {<der,{wff1,wff5}>}
  wff9!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
          {<hyp,{wff9}>}
  wff7!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
          {<hyp,{wff7}>}
  wff6!:  Location(belowGround)  {<der,{wff1,wff7}>}
  wff5!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
          {<hyp,{wff5}>}
  wff1!:  OnFloor(1)  {<hyp,{wff1}>}
```

```
: perform disbelieve(OnFloor(1))

: describe-context
((assertions (wff9 wff7 wff5)) (named (default-defaultct))
 (kinconsistent nil))

: list-asserted-wffs
  wff9!:  {OnFloor(4),OnFloor(3)} v=> {Location(aboveGround)}
          {<hyp,{wff9}>}
  wff7!:  {OnFloor(2),OnFloor(1)} v=> {Location(belowGround)}
          {<hyp,{wff7}>}
  wff5!:  xor{OnFloor(4),OnFloor(3),OnFloor(2),OnFloor(1)}
          {<hyp,{wff5}>}
```

SNePSLOG Example of $\neg I$

```
wff5!:  BearYoung(whales,live)  {<hyp,{wff5}>}
wff4!:  LiveIn(whales,water)  {<hyp,{wff4}>}
wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal))
        {<hyp,{wff3}>}
wff2!:  all(x)(LiveIn(x,water) => Ako(x,fish))
        {<hyp,{wff2}>}
wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
        {<hyp,{wff1}>}
```

```
: Ako(whales, ?x)?
  A contradiction was detected within context default-defaultct.
  The contradiction involves the newly derived proposition:
      wff8!:  Ako(whales,mammal)  {<der,{wff3,wff5}>}

  and the previously existing proposition:
      wff9!:  ~Ako(whales,mammal)  {<der,{wff1,wff2,wff4}>}

...
  In order to make the context consistent you must delete at least
  one hypothesis from each of the following sets of hypotheses:
    (wff5 wff4 wff3 wff2 wff1)
```

The Culprit Set

```
1 :   wff5!:  BearYoung(whales,live)  {<hyp,{wff5}>}
         (2 supported propositions: (wff8 wff5) )

2 :   wff4!:  LiveIn(whales,water)  {<hyp,{wff4}>}
         (3 supported propositions: (wff9 wff7 wff4) )

3 :   wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal)) {<hyp,{wff3
         (2 supported propositions: (wff8 wff3) )

4 :   wff2!:  all(x)(LiveIn(x,water) => Ako(x,fish)) {<hyp,{wff2}>}
         (3 supported propositions: (wff9 wff7 wff2) )

5 :   wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
              {<hyp,{wff1}>}
         (2 supported propositions: (wff9 wff1) )
```

KB after deleting `wff2`

```
wff10!:  ~(all(x)(LiveIn(x,water) => Ako(x,fish)))
         {<ext,{wff1,wff3,wff4,wff5}>}
wff8!:  Ako(whales,mammal)  {<der,{wff3,wff5}>}
wff7!:  ~Ako(whales,fish)  {<der,{wff1,wff3,wff5}>}
wff5!:  BearYoung(whales,live)  {<hyp,{wff5}>}
wff4!:  LiveIn(whales,water)  {<hyp,{wff4}>}
wff3!:  all(x)(BearYoung(x,live) => Ako(x,mammal))
        {<hyp,{wff3}>}
wff1!:  all(x)(nand{Ako(x,fish),Ako(x,mammal)})
        {<hyp,{wff1}>}
```

10 The Situation Calculus

Motivation (McCarthy)

I'm in my study at home. My car is in the garage. I want to get to the airport. How do I decide that I should walk to the garage and drive to the airport, rather than vice versa?

A commonsense planning problem.

Solution Sketch

My study and garage are in my home.
To get from one place to another in my home, I should walk.

My garage and the airport are in the county.
To get from one place to another in the county, I should drive.

Situations

When an agent acts, some propositions change as a result of acting, and some are independent of acting.

E.g. the fact that the airport is in the county is independent of my acting, but whether I'm in my study, in the garage, or at the airport, changes when I act.

We say that an act takes us from one situation to another.

Propositions that are dependent on situations are called propositional fluents. E.g. *At(study, S0), At(garage, S1)* vs. *In(study, home), In (airport, county)*

Situational Fluents

We can view an act as something that's done in some situation, and takes us to another situation.

Let *do(a, s)* be a two-argument functional term.

$[\![do(a, s)]\!]$ = the situation that results from doing the act $[\![a]\!]$ in the situation $[\![s]\!]$.

So, *At(study, S0)*, *At(garage, do(walk(study, garage), S0))*

Planning in the Situational Calculus

Describe the situation *S0*.

Give domain rules describing the effects of actions.

Find a solution for *At(airport, ?s)*

Formalization in SNARK
Non-Fluent Propositions

```
(assert '(Walkable home))
(assert '(Drivable county))
(assert '(In study home))
(assert '(In garage home))
(assert '(In garage county))
(assert '(In airport county))
```

Effect Axioms

```
(assert '(all (x y z s)
          (=> (and (At x s) (In x z) (In y z)
                   (Walkable z))
           (At y (do (walk x y) s)))))

(assert '(all (x y z s)
          (=> (and (At x s) (In x z) (In y z)
                   (Drivable z))
           (At y (do (drive x y) s)))))
```

Initial Situation

```
(assert '(At study S0))
```

SNARK Solves the Problem

```
(query "How do you go to the airport?"
       '(At airport ?s)
       :answer '(By doing ?s))

How do you go to the airport?
  (ask '(At airport ?s))
  = (At airport (do (drive garage airport)
                    (do (walk study garage) S0)))
```

Example 2: BlocksWorld Domain Axioms

```
(assert '(all s (Clear Table s)))

(assert '(all (x y s) (=> (and (Block y) (On x y s))
                           (not (Clear y s)))))

(assert '(all (x s) (=> (Held x s)
                         (not (Clear x s)))))
```

BlocksWorld Effect Axioms

```
(assert
    '(all (x y s) (=> (and (On x y s) (Clear x s))
                      (and (Held x (do (pickUp x) s))
                           (Clear y (do (pickUp x) s))))))
(assert
    '(all (x y s) (=> (and (Held x s) (Clear y s))
                      (and (On x y (do (putOn x y) s))
                           (not (Held x (do (putOn x y) s)))
                           (Clear x (do (putOn x y) s))))))
```

Initial Situation

```
(assert '(Block A))
(assert '(Block B))
(assert '(Block C))
(assert '(On A B S0))
(assert '(On B Table S0))
(assert '(On C Table S0))
(assert '(Clear A S0))
(assert '(Clear C S0))
```

Solving A Simple Problem

```
(query "How do you achieve holding Block A?"
       '(Held A ?s)
       :answer '(By doing ?s))

How do you achieve holding Block A?
  (ask '(Held A ?s)) = (By doing (do (pickUp A) S0))
```

A Harder Problem

```
(query "How do you put Block A on Block C"
       '(On A C ?s)
       :answer '(By doing ?s))
```

Just loops!

The Frame Problem

We want

`(On A C (do (putOn A C) (do (pickUp A) S0)))`

but this requires C to be clear in situation
`(do (pickUp A) S0)`

That can't be decided.

We need to specify what propositional fluents **don't change** when an action is performed.

A Frame Axiom

```
(assert
    '(all (x y s) (=> (and (Clear x s) (not (= x y)))
                      (Clear x (do (pickUp y) s)))))
```

Another Problem

Still doesn't work, because we don't know that

```
(not (= C A))
```

Unique Names Axioms

```
(assert '(not (= A B)))
(assert '(not (= A C)))
(assert '(not (= B C)))
```

Also need

```
(use-paramodulation)
```

after `(initialize)`

This includes the theory of equality with resolution.

Success!

```
(query "How do you put Block A on Block C" '(On A C ?s)
       :answer '(By doing ?s))

How do you put Block A on Block C
  (ask '(On A C ?s))
  = (By doing (do (putOn A C) (do (pickUp A) S0)))
```

11 Summary

Artificial Intelligence (AI): A field of computer science and engineering concerned with the computational understanding of what is commonly called intelligent behavior, and with the creation of artifacts that exhibit such behavior.

Knowledge Representation and Reasoning (KR or KRR): A subarea of Artificial Intelligence concerned with understanding, designing, and implementing ways of representing information in computers, and using that information to derive new information based on it.

KR is more concerned with belief than "knowledge". Given that an agent (human or computer) has certain beliefs, what else is reasonable for it to believe, and how is it reasonable for it to act, regardless of whether those beliefs are true and justified.

What is Logic?

- **Logic** is the study of correct reasoning.
- There are many systems of logic (logics). Each is specified by specifying:
  - Syntax: Specifying what counts as a well-formed expression
  - Semantics: Specifying the meaning of well-formed expressions
    - Intensional Semantics: Meaning relative to a Domain
    - Extensional Semantics: Meaning relative to a Situation
  - Proof Theory: Defining proof/derivation, and how it can be extended.

KR and Logic

Given that a Knowledge Base is represented in a language with a well-defined syntax, a well-defined semantics, and that reasoning over it is a well-defined procedure, a KR system is a logic.

KR research can be seen as a search for the best logic to capture human-level reasoning.

Proof Theory and Semantics

Proof	**Derivation**		**Theoremhood**
Theory	$A_1, \ldots, A_n \vdash P$	$\Leftrightarrow$	$\vdash A_1 \land \ldots \land A_n \Rightarrow P$
	$\Downarrow\Uparrow$		$\Downarrow\Uparrow$
	$A_1, \ldots, A_n \models P$	$\Leftrightarrow$	$\models A_1 \land \ldots \land A_n \Rightarrow P$
Semantics	**Logical Implication**		**Validity**
	($\Downarrow$ *Soundness*)		($\Uparrow$ *Completeness*)

Inference/Reasoning Methods

Given a KB/set of assumptions $\mathcal{A}$ and a query $\mathcal{Q}$:

- Model Finding
  - Direct: Find satisfying models of $\mathcal{A}$;
    see if $\mathcal{Q}$ is true in all of them.
  - Refutation: Find if $\mathcal{A} \cup \{\neg\mathcal{Q}\}$ is unsatisfiable.
- Natural Deduction
  - Direct: Find if $\mathcal{A} \vdash \mathcal{Q}$.
- Resolution
  - Direct: Find if $\mathcal{A} \vdash \mathcal{Q}$ (incomplete).
  - Refutation: Find if $\bigwedge \mathcal{A} \wedge \neg\mathcal{Q}$ is inconsistent.

Logics We Studied

1. Standard Propositional Logic
2. Clause Form Propositional Logic
3. Standard Finite-Model Predicate Logic
4. Clause Form Finite-Model Predicate Logic
5. Standard First-Order Predicate Logic
6. Clause Form First-Order Predicate Logic
7. Horn Clause Logic
8. Relevance Logic
9. SNePSLOG & SNeRE
10. The Situation Calculus
11. Description Logics

Classes of Logics

- Propositional Logic
  - Finite number of atomic propositions and models.
  - Model finding and resolution are decision procedures.
- Finite-Model Predicate Logic
  - Finite number of terms, atomic formulae, and models.
  - Reducible to propositional logic.
  - Model finding and resolution are decision procedures.
- First-Order Logic
  - Infinite number of terms, atomic formulae, and models.
  - Not reducible to propositional logic.
  - There are no decision procedures.
  - Resolution plus factoring is refutation complete.

Proof Procedures We Studied

1. Direct model finding: truth tables, `decreasoner`, `relsat` (complete search) `walksat, gsat` (stochastic search)
2. Wang algorithm (model-finding refutation), `wang`
3. Semantic tableaux (model-finding refutation)
4. Hilbert-style axiomatic (direct), *brief*
5. Fitch-style natural deduction (direct)
6. Resolution (refutation), `prover`, `SNARK`
7. SLD resolution (refutation), `Prolog`
8. SNePS (direct), `SNePS`

Utility Notions and Techniques

1. Material implication
2. Possible properties of connectives
   - commutative, associative, idempotent
3. Possible properties of well-formed expressions
   - free, bound variables
   - open, closed, ground expressions
4. Possible semantic properties of wffs
   - contradictory, satisfiable, contingent, valid
5. Possible properties of proof procedures
   - sound, consistent, complete,
   - decision procedure, semi-decision procedure

More Utility Notions and Techniques

5. Substitutions

    application, composition

6. Unification

    most general common instance (mgci),
    most general unifier (mgu)

7. Translation from standard form to clause form

    Conjunctive Normal Form (CNF),
    Skolem functions/constants

8. Resolution Strategies

    subsumption, unit preference, set of support

9. The Answer Literal

Yet More Utility Notions and Techniques

9. Closed *vs.* Open World Assumption
10. Negation by failure
11. Origin sets, contexts
12. Belief Revision/Truth-Maintenance

Domain Modeling

1. Formalization in various logics
2. Reification
3. Ontologies/Taxonomies/Hierarchies
    - extensional *vs.* intensional
    - instance *vs.* subcategory
    - Single (DAGs) vs multiple inheritance
    - transitive relations/transitive closure
    - mutually exclusive/disjoint categories
    - exhaustive set of subcategories
    - partitioning of a category

More Domain Modeling

4. Time
   - subjective *vs.* objective
   - points *vs.* intervals
   - Allen's relations
5. Things (Count Nouns) *vs.* Substances (Mass Nouns)
6. Acting
   - situations
   - fluents

12 Production Systems
Architecture
Working (Short-term) Memory

Contains set (unordered, no repeats) of
Working Memory Elements (WMEs).
Each being a rather flat, ground (no variables) symbol structure.

Rule (Long-term) Memory

Contains set (unordered, no repeats) of
Production Rules.
Each being a condition-action rule
of form
if condition$_1$ …condition$_n$ **then** action$_1$ …action$_m$
Each condition and action being like a WME,
but allowing variables (and, maybe, other expressions)

Rule Triggering

A rule **if** $\text{condition}_1 \ldots \text{condition}_n$ **then** $\text{action}_1 \ldots \text{action}_m$
is triggered
if there is a substitution, σ
such that each $\text{condition}_i\sigma$ is a WME.

A single rule can be triggered in multiple ways (by multiple substitutions).

Rule Firing

A rule **if** condition$_1$ $\ldots$condition$_n$ **then** action$_1$ $\ldots$action$_m$
that is triggered in a substitution σ
fires by performing every action$_i\sigma$.

Production System Execution Cycle

```
loop
    Collect $\mathcal{T}$= {r$\sigma$ | r$\sigma$ is a triggered rule}
    if $\mathcal{T}$ is not empty
        Choose a $r\sigma \in \mathcal{T}$
        Fire $r\sigma$
    until $\mathcal{T}$ is empty.
```

Some Typical Actions

- **stop**
- **delete** a WME
- **add** a WME
- **modify** a WME
- formatted print

Conflict Resolution Strategies

Purpose: to "Choose a $r\sigma \in \mathcal{T}$"

Specificity: If the conditions of one rule are a subset of a second rule, choose the second rule. [B & L, p. 126]

Recency: Based on recency of addition or modification of WMEs, or on recency of a rule firing. [B & L, p. 126]

Refactoriness: Don't allow the same substitution instance of a rule to fire again. [B & L, p. 127]

Salience: Explicit salience value. "The use of salience is generally discouraged" [`http://herzberg.ca.sandia.gov/jess/docs/70/rules.html#salience`].

The Rete Algorithm Assumptions

Rule memory doesn't change.

WM changes only slightly on each cycle.

WMEs are ground.

Production Systems are data-driven (use forward chaining).

Many rules share conditions.

The Rete Network

Create a network from the conditions (Like a discrimination tree) with rules at the leaves.

Create a token for each WME.

Pass each token through the network, stopping when it doesn't satisfy a test; resuming when the WME is modified.

When tokens reach a leaf, the rule is triggered.

Kinds of branch nodes

α *nodes:* Simple test.
β *nodes:* Constraints caused by different conditions.

13 Description Logics

Main reference:

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation and Applications, Second Edition,* Cambridge University Press, Cambridge, UK, 2007.

DL: Main Ideas

- Terminological Box or T-Box.
  Definition of *Concepts* ("Classes") and *Roles* ("Properties").
- Assertional Box or A-Box.
  Assertions about individuals (instances)
  - Unary predicates = concepts
  - Binary predicates = roles
- Necessary and Sufficient conditions on classes.
- Subsumption Hierarchy

Syntax of a Simple DL[a]
Atomic Symbols

- Positive integers: 1, 2, 3
- Atomic concepts: Thing, Pizza, PizzaTopping, PizzaBase
  Thing is the top of the hierarchy.
- Roles: hasTopping, hasBase
- Constants: item1, item2

---

[a] From Ronald J. Brachman & Hector J. Levesque, *Knowledge Representation and Reasoning,* Morgan Kaufmann/Elsevier, 2004, Chapter 9, with examples from Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, & Chris Wroe, A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools: Edition 1.1, The University of Manchester, 2007.

Syntax of a Simple DL Concepts

- Every atomic concept is a concept
- If r is a role and d is a concept, [ALL r d] is a concept.
  The concept of individuals all of whose r's are d's.
  E.g., [ALL hasTopping VegetarianTopping]
- If r is a role and n is a positive integer, [EXISTS n r] is a concept.
  The concept of individuals that have at least n r's.
  E.g., [EXISTS 1 hasTopping]
- If r is a role and c is a constant, [FILLS r c] is a concept.
  The concept of individuals one of whose r's is c.
  E.g., [FILLS hasTopping item2]
- If $d_1, \ldots, d_n$ are concepts, [AND $d_1, \ldots, d_n$] is a concept
  The concept that is the intersection of $d_1, \ldots, d_n$.
  E.g., [AND Pizza [EXISTS 1 hasTopping]
  [ALL hasTopping VegetarianTopping]]

Syntax of a Simple DL
Sentences

- If d_1 and d_2 are concepts, $(d_1 \sqsubseteq d_2)$ is a sentence.
  d_1 is subsumed by d_2
  E.g., VegetarianPizza $\sqsubseteq$ Pizza
- If d_1 and d_2 are concepts, $(d_1 \doteq d_2)$ is a sentence.
  d_1 and d_2 are equivalent
  E.g., VegetarianPizza $\doteq$ [AND Pizza [EXISTS 1 hasTopping] [ALL hasTopping VegetarianTopping]]
- If c is a constant and d is a concept, $(c \rightarrow d)$ is a sentence.
  The individual c satisfies the description expressed by d.
  E.g., item1 → Pizza

Necessary and Sufficient Conditions

A **necessary** condition on a class, d, is a property, p, such that if an individual, c, is an instance of d, it is **necessary** that c satisfy p.

A **sufficient** condition on a class, d, is a property, p, such that if an individual, c, satisfies p, then that is a **sufficient** reason to decide that it is an instance of d.

A **defined** concept has both necessary and sufficient conditions.

A **primitive** concept has only necessary conditions.

Subsumption Hierarchy

$(d_1 \sqsubseteq d_2)$

d_1 is subsumed by d_2

E.g., VegetarianPizza $\sqsubseteq$ Pizza

means that every instance of d_1 is an instance of d_2.

Every DL concept is subsumed by Thing, the top of the hierarchy.

Classification Algorithm

Decision procedure for placing every defined concept correctly in the subsumption hierarchy.

Note: Two concepts that subsume each other are the same.

Note: No concept can be computed as being subsumed by a primitive concept.

Examples Using Classic
Defined and Primitive Concepts

```
: (cl-startup)
t

: (cl-define-concept ’PizzaTopping ’Classic-Thing)
*WARNING*: The new concept PizzaTopping is identical
           to the existing concept @c{Classic-Thing}.
@c{Classic-Thing}

: (cl-define-primitive-concept ’PizzaBase ’Classic-Thing)
@c{PizzaBase}
```

Creating An Individual

```
: (cl-create-ind 'base1 'PizzaBase)
@i{base1}

: (cl-instance? @base1 @PizzaBase)
t

: (cl-print-ind @base1)
Base1 ->
 Derived Information:
  Primitive ancestors: PizzaBase Classic-Thing
  Parents: PizzaBase
  Ancestors: Thing Classic-Thing
@i{base1}
```

Defining Some Roles

```
: (cl-define-primitive-role ’hasIngredient
                            :inverse ’isIngredientOf)
@r{hasIngredient}

: (cl-define-primitive-role ’hasBase :parent ’hasIngredient
                            :inverse ’isBaseOf)
@r{hasBase}

: (cl-define-primitive-role ’hasTopping :parent ’hasIngredient
                            :inverse ’isToppingOf)
@r{hasTopping}
```

Necessary and Sufficient Conditions

```
: (cl-define-concept 'Pizza '(and Classic-Thing (at-least 1 hasBase)
                                                (at-least 1 hasTopping)))
@c{Pizza}
: (cl-create-ind 'pizza1 'Pizza)
@i{pizza1}
: (cl-print-ind @pizza1)
Pizza1 ->
 Derived Information:
  Parents: Pizza
  Ancestors: Thing Classic-Thing
  Role Fillers and Restrictions:
   Hasingredient[1 ; INF]
   Hastopping[1 ; INF]
   Hasbase[1 ; INF]
@i{pizza1}
: (cl-create-ind 'item3 '(and (fills hasBase base3) (fills hasTopping topping3)))
@i{item3}
: (cl-print-ind @item3)
Item3 ->
 Derived Information:
  Parents: Pizza
  Ancestors: Thing Classic-Thing
  Role Fillers and Restrictions:
   Hasingredient[2 ; INF] -> Base3 Topping3
   Hastopping[1 ; INF] -> Topping3
   Hasbase[1 ; INF] -> Base3
@i{item3}
```

Classification

```
: (cl-define-concept 'PreparedFood '(and Classic-Thing (at-least 1 hasIngredient)))
@c{PreparedFood}

: (cl-print-concept @PreparedFood)
PreparedFood ->
 Derived Information:
  Parents: Classic-Thing
  Ancestors: Thing
  Children: Pizza
  Role Restrictions:
   Hasingredient[1 ; INF]
@c{PreparedFood}

: (cl-print-concept @Pizza)
Pizza ->
 Derived Information:
  Parents: PreparedFood
  Ancestors: Thing Classic-Thing
  Role Restrictions:
   Hasingredient[1 ; INF]
   Hastopping[1 ; INF]
   Hasbase[1 ; INF]
@c{Pizza}

: (cl-instance? @pizza1 @PreparedFood)
t
```

Disjoint Concepts

```
: (cl-startup)
t
: (cl-define-primitive-concept 'PizzaTopping 'Classic-Thing)
@c{PizzaTopping}
: (cl-define-disjoint-primitive-concept 'CheeseTopping 'PizzaTopping 'pizzaToppings)
@c{CheeseTopping}
: (cl-define-disjoint-primitive-concept 'MeatTopping 'PizzaTopping 'pizzaToppings)
@c{MeatTopping}
: (cl-define-disjoint-primitive-concept 'SeafoodTopping 'PizzaTopping 'pizzaToppings)
@c{SeafoodTopping}
: (cl-define-disjoint-primitive-concept 'VegetableTopping 'PizzaTopping 'pizzaToppings)
@c{VegetableTopping}
classic(56): (cl-define-primitive-concept 'ProbeInconsistentTopping
                                          '(and CheeseTopping VegetableTopping))
*WARNING*: Disjoint primitives: @tc{CheeseTopping}, @tc{VegetableTopping}.
*CLASSIC ERROR* while processing
  (cl-define-primitive-concept ProbeInconsistentTopping (and CheeseTopping
    VegetableTopping))
    occurred on object @c{ProbeInconsistentTopping-*INCOHERENT*}:
  Trying to combine disjoint primitives: @tc{CheeseTopping} and
    @tc{VegetableTopping}.
classic-error
(disjoint-prims-conflict @tc{CheeseTopping} @tc{VegetableTopping})
nil
@c{ProbeInconsistentTopping-*INCOHERENT*}
```

Open World

```
: (cl-define-primitive-concept 'MushroomTopping 'VegetableTopping)
@c{MushroomTopping}
: (cl-define-primitive-concept 'OnionTopping 'VegetableTopping)
@c{OnionTopping}
: (cl-define-concept 'VegetarianPizza '(and Pizza (all hasTopping VegetableTopping)))
@c{VegetarianPizza}

: (cl-create-ind 'mt1 'MushroomTopping)
@i{mt1}
: (cl-create-ind 'ot1 'OnionTopping)
@i{ot1}
: (cl-create-ind 'pizza2 '(and Pizza (fills hasTopping mt1) (fills hasTopping ot1)))
@i{pizza2}

: (cl-instance? @pizza2 @VegetarianPizza)
nil
: (cl-ind-close-role @pizza2 @hasTopping)
@i{pizza2}
: (cl-instance? @pizza2 @VegetarianPizza)
t
```

Typology of DL Languages

<table>
<tr><th>Construct</th><th>Syntax</th><th colspan="4">Language</th></tr>
<tr><td>Concept</td><td>A</td><td rowspan="4">FL_0</td><td rowspan="4">FL^-</td><td rowspan="7">AL</td><td rowspan="11">S</td></tr>
<tr><td>Role name</td><td>R</td></tr>
<tr><td>Intersection</td><td>$C \cap D$</td></tr>
<tr><td>Value Restriction</td><td>$\forall R.C$</td></tr>
<tr><td>Limited existential quantification</td><td>$\exists R.\top$</td><td colspan="2" rowspan="3"></td></tr>
<tr><td>Top or Universal</td><td>$\top$</td></tr>
<tr><td>Bottom</td><td>$\bot$</td></tr>
<tr><td>Atomic negation</td><td>$\neg A$</td><td colspan="3"></td></tr>
<tr><td>Negation</td><td>$\neg C$</td><td colspan="3">C</td></tr>
<tr><td>Union</td><td>$C \cup D$</td><td colspan="3">U</td></tr>
<tr><td>Existential restriction</td><td>$\exists R.C$</td><td colspan="3">E</td></tr>
</table>

Language S = ALC$_{R+}$ = ALC plus transitive roles.

From A. Gòmez-Pèrez, M. Fernàndez-Lòpez & O. Corcho, *Ontological Engineering,* Springer-Verlag, London, 2004, Table 1.1, p. 17.

Typology, continued

Construct	**Syntax**	**Language**
Number restrictions	$(\geq \mathrm{n}\ \mathrm{R})\ (\leq \mathrm{n}\ \mathrm{R})$	N
Nominals	$\{\mathrm{a}_1 \ldots \mathrm{a}_\mathrm{n}\}$	O
Role hierarchy	$\mathrm{R} \subseteq \mathrm{S}$	H
Inverse role	R'	I
Qualified number restriction	$(\geq \mathrm{n}\ \mathrm{R.C})\ (\leq \mathrm{n}\ \mathrm{R.C})$	Q

Key to abbreviations under "Syntax":

A: atomic concept

C, D: concept definitions

R: atomic role

S: role definition

From A. Gòmez-Pèrez, M. Fernàndez-Lòpez & O. Corcho, *Ontological Engineering,* Springer-Verlag, London, 2004, Table 1.1, p. 17.

14 Abduction

Abduction is the non-sound inference

from

$P \Rightarrow Q$

and Q

to

P

See *Brachman & Levesque,* Chapter 13.

Some Uses of Abduction

1. Explanation

   from $It's\ raining \Rightarrow The\ grass\ is\ wet$

   and $The\ grass\ is\ wet$ to $It's\ raining$

2. Diagnosis

   from $Infection \Rightarrow Fever$

   and $Fever$ to $Infection$

3. Plan Recognition

   from $Cooking\ pasta \Rightarrow Boil\ water$

   and $Boil\ water$ to $Cooking\ pasta$

4. Text Understanding

   from $\forall x(gotGoodService(x) \Rightarrow leftBigTip(x))$

   and *Betty left a big tip.* to *Betty got good service.*

Prime Implicates

Applies to KRR using resolution.

For some KB and some clause C, if

KB $\models C$

and for any $C\prime$ s.t. $C\prime$ is a proper subset of C

KB $\not\models C\prime$

C is a prime implicate of KB.

Example of Computing Prime Implicate

```
prover(4): (prove '( (=> (and p q r) g)
                     (=> (and (not p) q) g)
                     (=> (and (not q) r) g))
                  'g)

 1  (p (not q) g)                          Assumption
 2  (q (not r) g)                          Assumption
 3  ((not p) (not q) (not r) g)            Assumption
 4  ((not g))                              From Query

 5  (p (not q))                   R,4,1,{}
 6  (q (not r))                   R,4,2,{} Subsumed
 7  ((not p) (not q) (not r)) R,4,3,{} Subsumed
 8  ((not r) p)                   R,5,6,{} Subsumed
11  ((not q) (not r))             R,7,8,{} Subsumed
12  ((not r))                     R,11,6,{}
```

Example from *Brachman & Levesque,* p 271.

Example 2

```
prover(8): (prove '((forall x (=> (enterRestaurant x) (beSeated x)))
                    (forall x (=> (beSeated x) (beServed x)))
                    (forall x (=> (beServed x) (getFood x)))
                    (forall x (=> (getFood x) (eatFood x)))
                    (forall x (=> (eatFood x) (and (pay x) (leaveTip x))))
                    (forall x (=> (gotGoodService x) (leftBigTip x)))
                    (enterRestaurant Betty))
                  '(leftBigTip Betty))

 1  ((enterRestaurant Betty)) Assumption
 2  ((not (enterRestaurant ?1)) (beSeated ?1))  Assumption
 3  ((not (beSeated ?3)) (beServed ?3)) Assumption
 4  ((not (beServed ?5)) (getFood ?5))  Assumption
 5  ((not (getFood ?7)) (eatFood ?7)) Assumption
 6  ((not (eatFood ?9)) (pay ?9)) Assumption
 7  ((not (eatFood ?10)) (leaveTip ?10))  Assumption
 8  ((not (gotGoodService ?12)) (leftBigTip ?12)) Assumption
 9  ((not (leftBigTip Betty)) (Answer (leftBigTip Betty)))  From Query
10  ((not (gotGoodService Betty))
     (Answer (leftBigTip Betty)))  R,9,8,{Betty/?12}
nil
```

I.e., `(=> (gotGoodService Betty) (leftBigTip Betty))`

Interpretation

Possible interpretations of
`(=> (gotGoodService Betty) (leftBigTip Betty))`:

1. Abduction: Since `(leftBigTip Betty)`,
   infer `(gotGoodService Betty)`.
2. Diagnosis: Since `(not (leftBigTip Betty))`,
   infer `(not (gotGoodService Betty))`.
3. Hypothetical Answer: If `(gotGoodService Betty)`
   then `(leftBigTip Betty)`.
4. Why Not: Didn't infer `(leftBigTip Betty)`
   because didn't know `(gotGoodService Betty)`.