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

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.

	Betty drives Tom.	Tom drives Betty.
Propositions:	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

$\neg$: $\sim$ $\texttt{!}$

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

$\vee$: $|$

$\Rightarrow$: $\rightarrow$ $\supset$ $\texttt{->}$

$\Leftrightarrow$: $\leftrightarrow$ $\equiv$ $\texttt{<->}$

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)`
$P \land Q$	`(and P Q)`
$P \lor Q$	`(or P Q)`
$P \Rightarrow Q$	`(if P Q)`
$P \Leftrightarrow Q$	`(iff P Q)`
$(P_1 \land \cdots \land P_n)$	`(and P1 ...Pn)`
$(P_1 \lor \cdots \lor P_n)$	`(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

[*Betty drives Tom*] = Tom drives Betty to work.

[*Tom drives Betty*] = Betty drives Tom to work.

[*Betty is the driver*] = Tom is the passenger in the car.

[*Tom is the driver*] = Betty is the passenger in the car.

[*Betty is the passenger*] = Tom is the driver of the car.

[*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

$[\textit{Betty drives Tom} \Leftrightarrow \neg \textit{Tom is the driver}]$
$=$ 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 $P \lor Q$ 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]

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

$$T : TD \Rightarrow BP$$

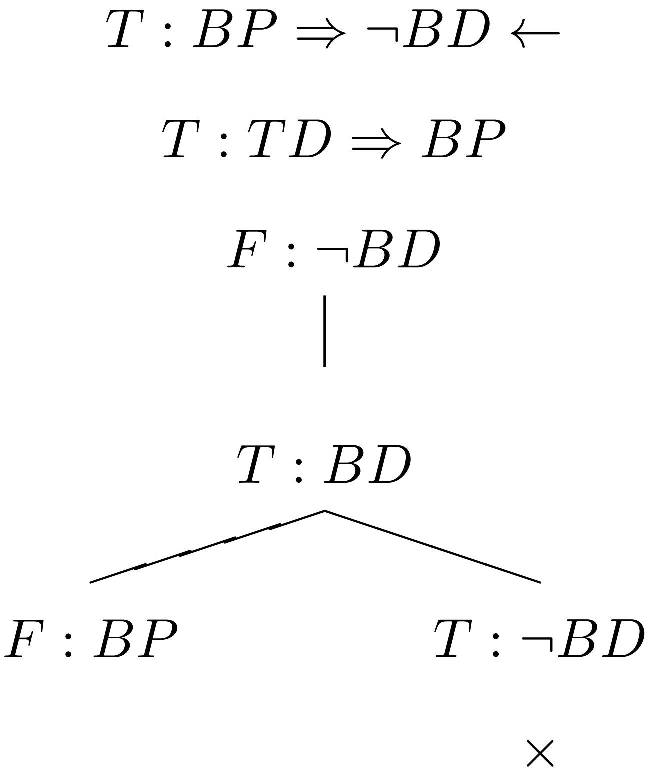
$$F : \neg BD$$

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

Tableau Procedure Example: Step 1

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

Tableau Procedure Example: Step 2



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

$T : TD \Rightarrow BP$

$F : \neg BD$

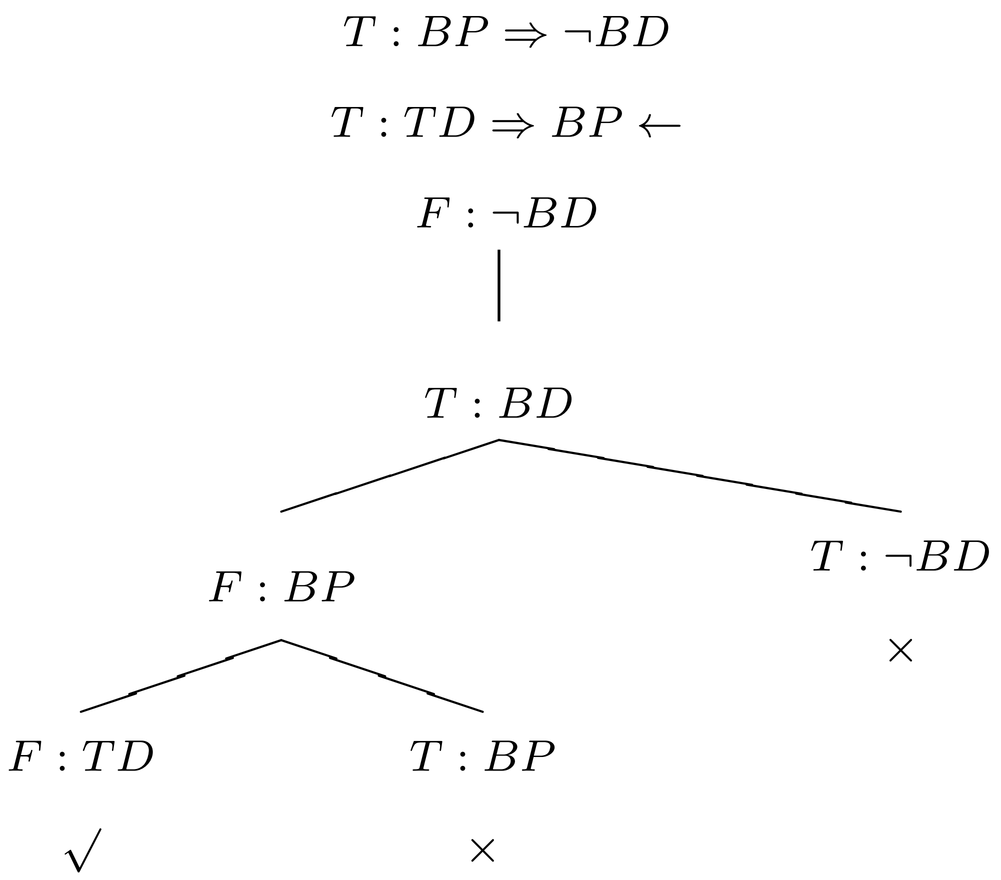
$T : BD$

$F : BP$

$T : \neg BD$

$\times$

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: $[\![BD]\!] =$ True; $[\![BP]\!] =$ False; $[\![TD]\!] =$ 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

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

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

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

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

$$Tom\ drives\ Betty \lor Betty\ drives\ Tom$$

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
Betty drives Tom ⇔ ¬*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 $TD, TD \Rightarrow BP, BP \Rightarrow \neg BD \models \neg BD$

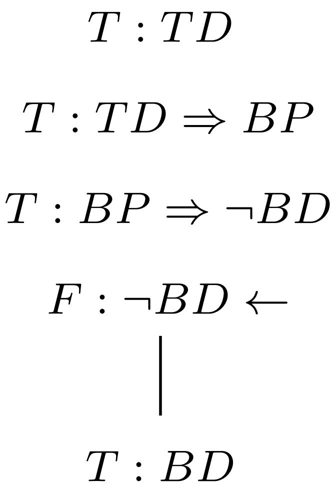
$$T : TD$$

$$T : TD \Rightarrow BP$$

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

$$F : \neg BD$$

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



$T : TD$

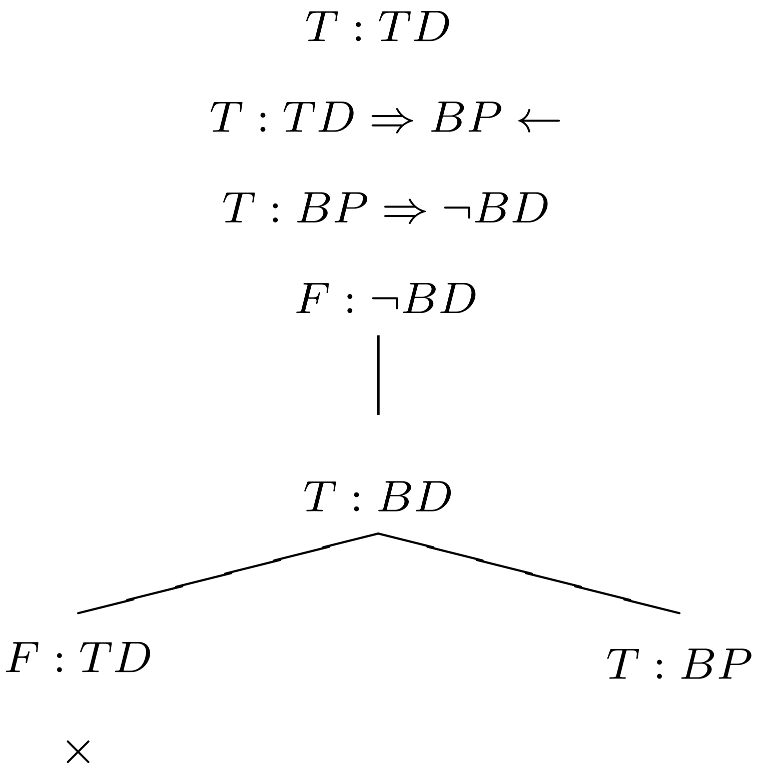
$T : TD \Rightarrow BP$

$T : BP \Rightarrow \neg BD$

$F : \neg BD \leftarrow$

$T : BD$

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



$T : TD$

$T : TD \Rightarrow BP \leftarrow$

$T : BP \Rightarrow \neg BD$

$F : \neg BD$

$T : BD$

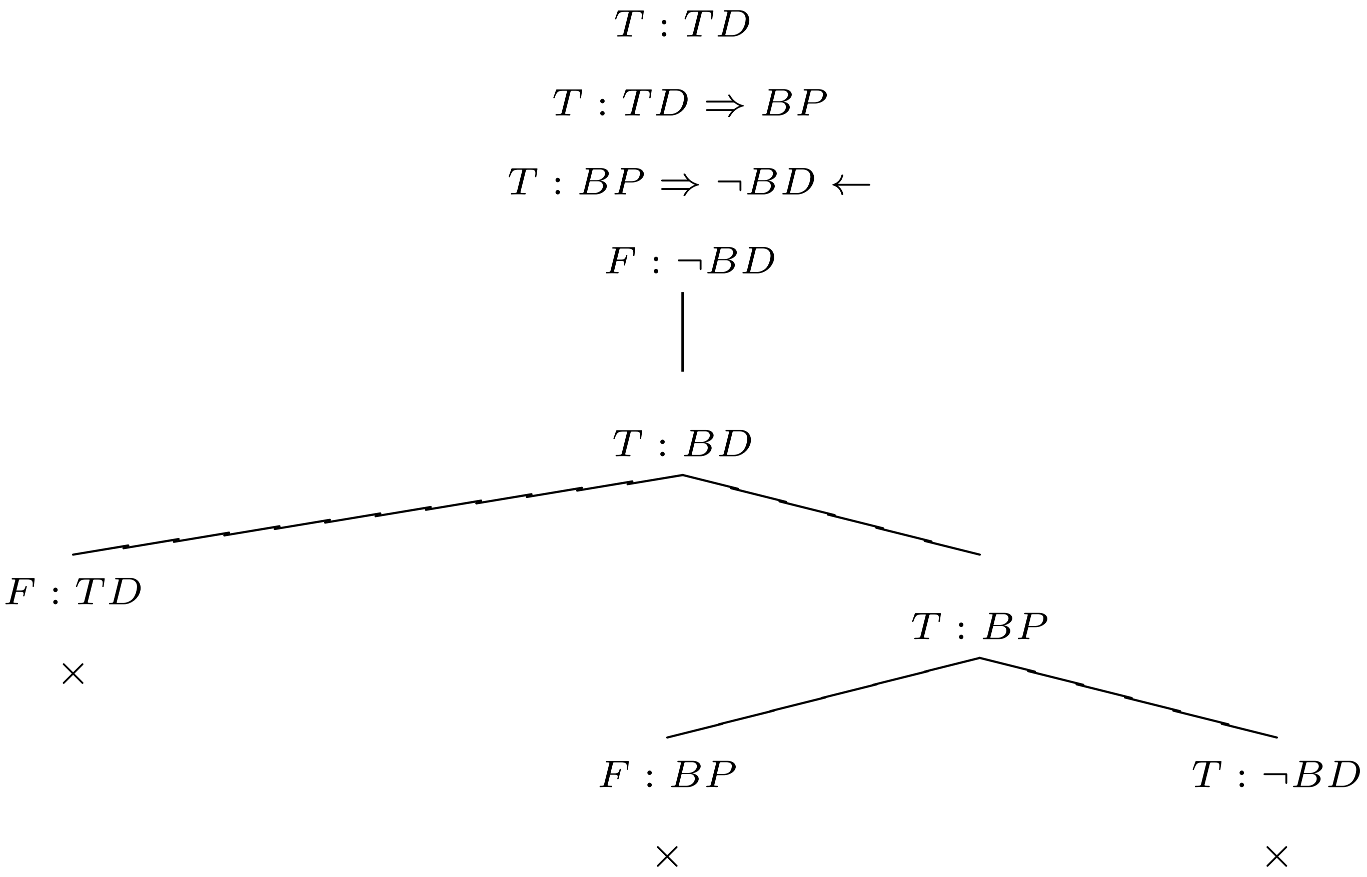
$F : TD$

$T : BP$

×

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 $TD \Rightarrow BP, BP \Rightarrow \neg BD \not\models \neg BD$

$$T : TD \Rightarrow BP$$

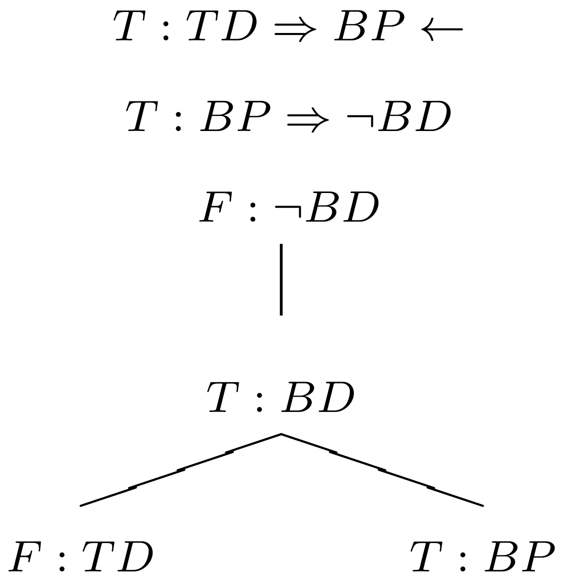
$$T : BP \Rightarrow \neg BD$$

$$F : \neg BD$$

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

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

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



$T : TD \Rightarrow BP \leftarrow$

$T : BP \Rightarrow \neg BD$

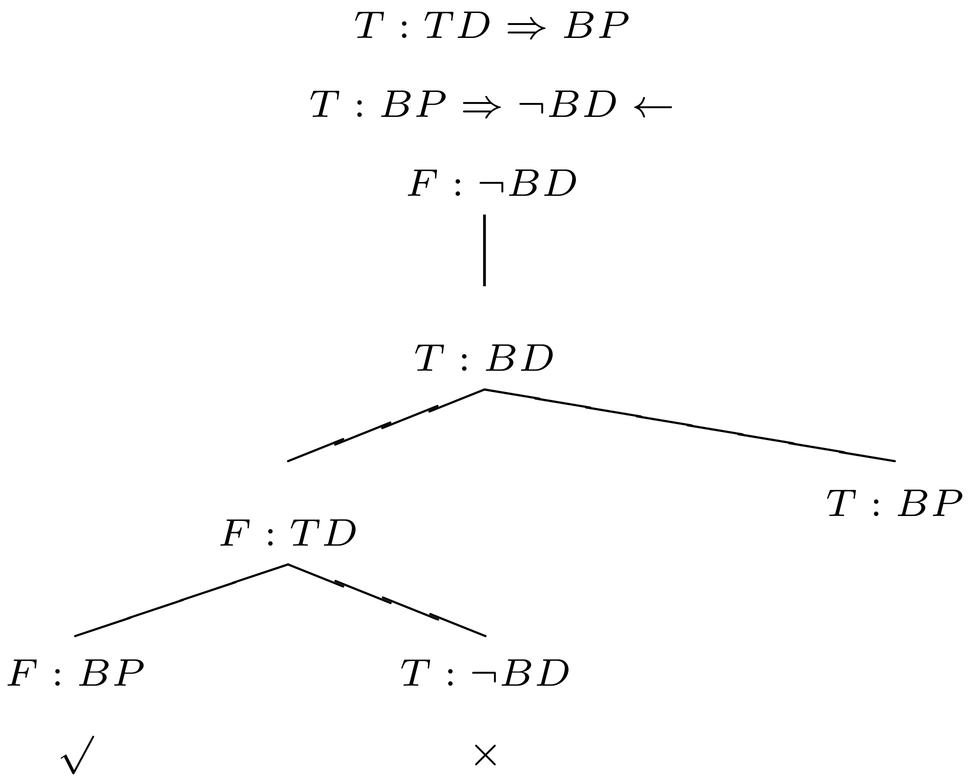
$F : \neg BD$

$T : BD$

$F : TD$

$T : BP$

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

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

$$Tom\ is\ the\ driver$$

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

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

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

(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]

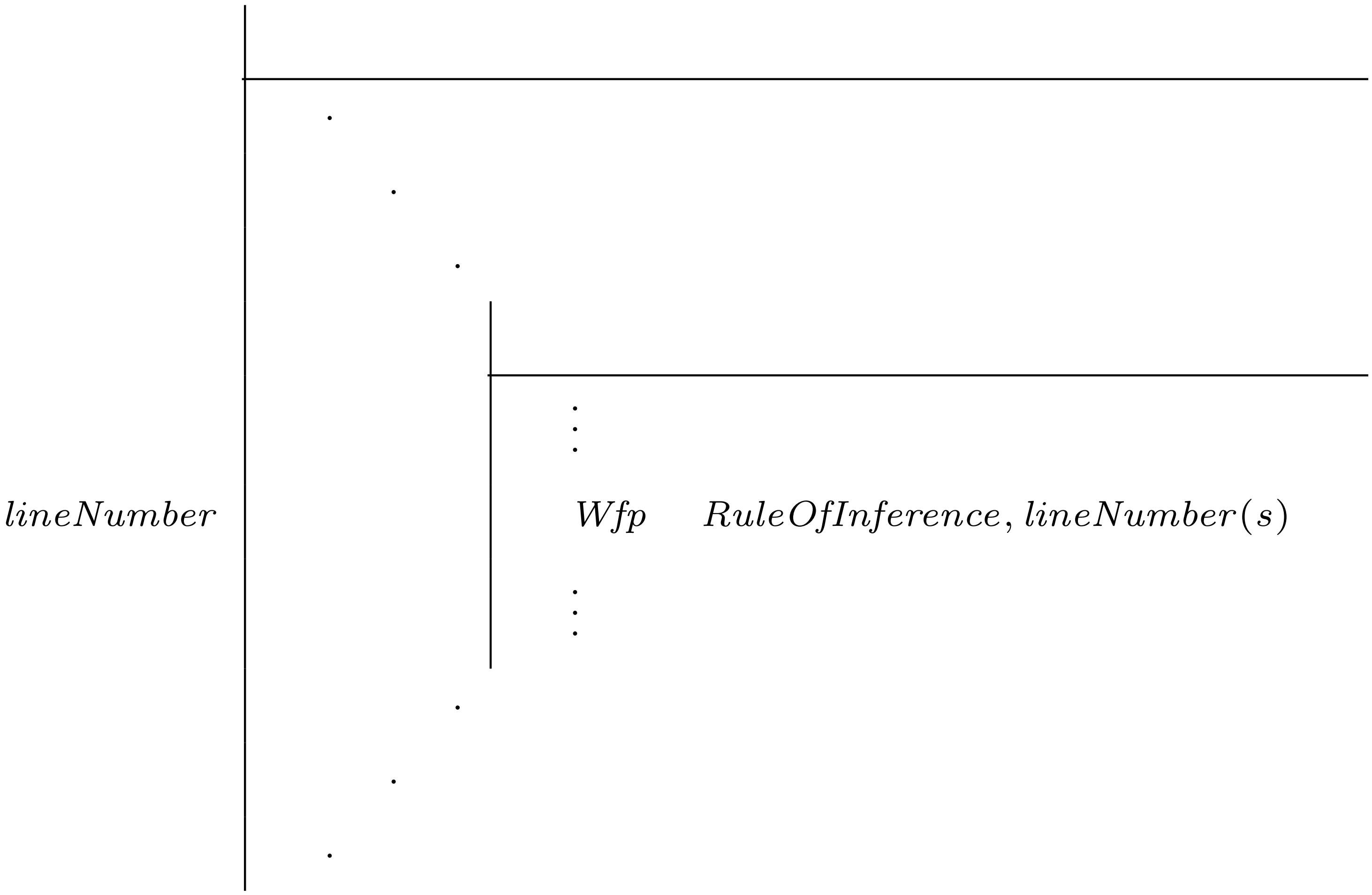
- A **proof** of a theorem P is
  - An ordered list of wfps and subproofs ending with P
  - Each wfp or subproof on the list must be justified by a rule of inference.
- $\vdash P$ is read "P is a theorem."

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

Fitch-Style Derivation

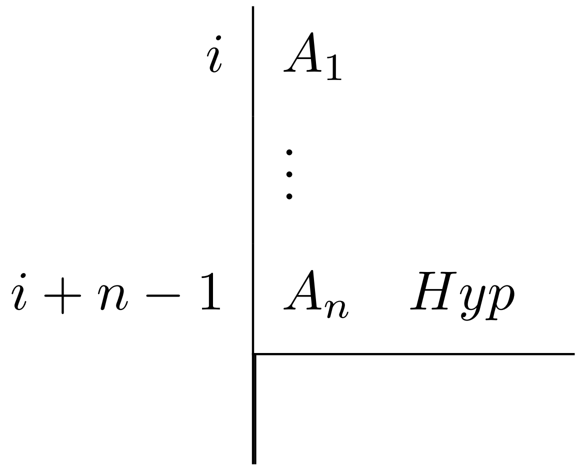
- A **derivation** of a wfp P from an assumption A is a hypothetical subproof whose hypothesis is A and which contains
  - An ordered list of wfps and inner subproofs ending with P
  - Each wfp or inner subproof on the list must be justified by a rule of inference.
- $A \vdash P$ is read "P can be derived 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 *Wfp* $\quad$ *RuleOfInference*, *lineNumber*(*s*)

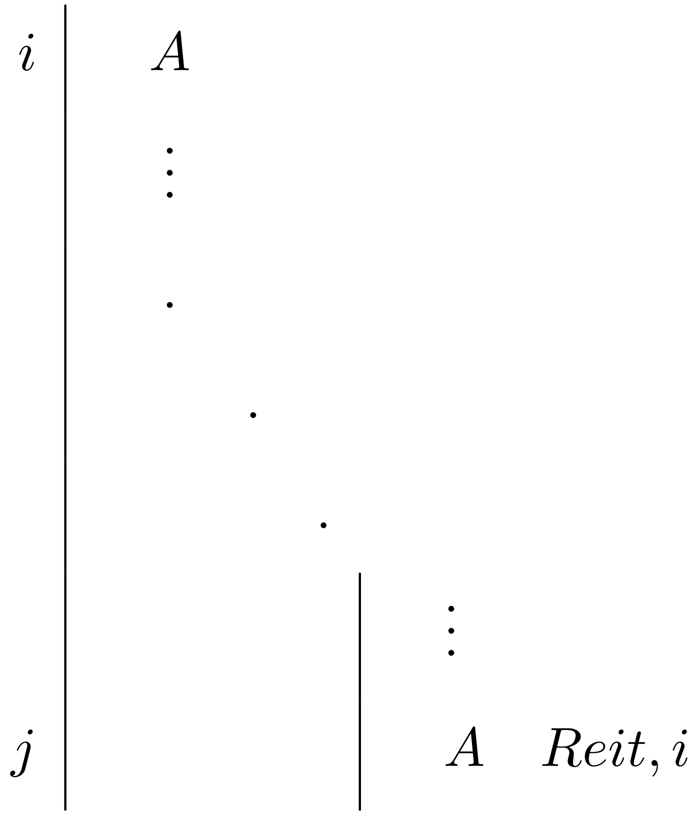
Structural Rules of Inference



i A_1

$\vdots$

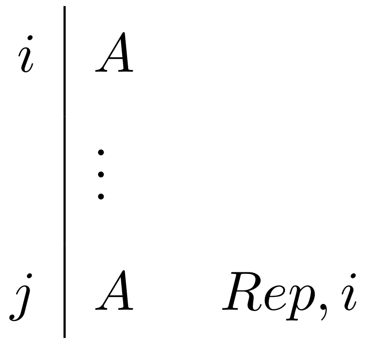
$i+n-1$ A_n Hyp



i A

$\vdots$

j A $Reit, i$



i A

$\vdots$

j A Rep, i

Rules for ⇒

$$\begin{array}{r|l}
i & \begin{array}{|cc} A_1 & \\ \vdots & \\ A_n & Hyp \\ \hline \vdots & \\ B & \end{array} \\
k & (A_1 \wedge \ldots \wedge A_n) \Rightarrow B \quad \Rightarrow I, i\text{–}j
\end{array}$$

Line labels of the left proof: i (A_1), $i+n-1$ (A_n), j (B), k.

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

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

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

Fitch-Style Proof of Axiom A1

$$
\begin{array}{lllr}
1. & \mid & \underline{\qquad\quad A \qquad\quad} & Hyp \\
2. & \mid & \mid \; \underline{\qquad B \qquad} & Hyp \\
3. & \mid & \mid \qquad A & Reit, 1 \\
4. & \mid & \quad B \Rightarrow A & \Rightarrow I, 2\text{–}3 \\
5. & & A \Rightarrow (B \Rightarrow A) & \Rightarrow I, 1\text{–}4
\end{array}
$$

Example Fitch-Style Derivation

that

$$Tom\ is\ the\ driver$$

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

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

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

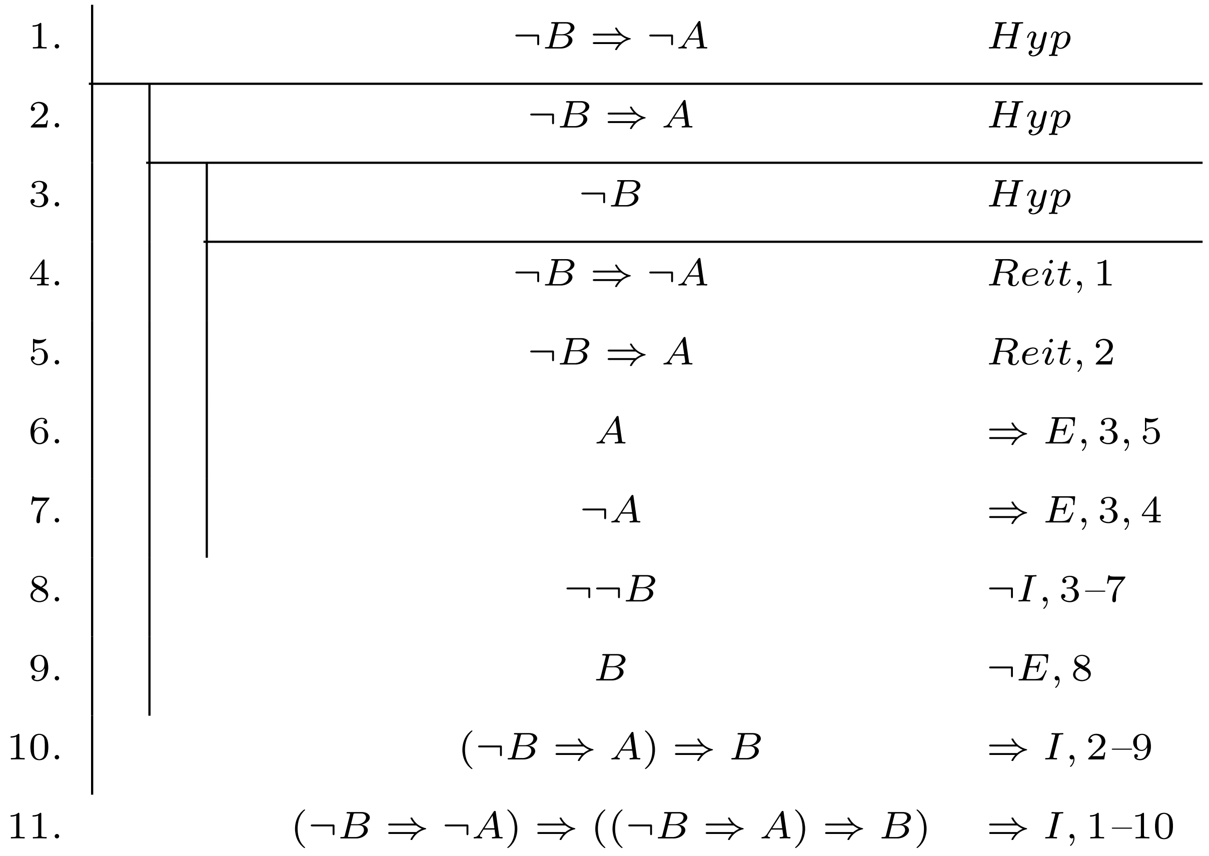
$$\begin{array}{r|ll}
1. & Tom\ is\ the\ driver & \\
2. & Tom\ is\ the\ driver \Rightarrow Betty\ is\ the\ passenger & \\
3. & Betty\ is\ the\ passenger \Rightarrow \neg Betty\ is\ the\ driver & Hyp \\
\hline
4. & Betty\ is\ the\ passenger & \Rightarrow E, 1, 2 \\
5. & \neg Betty\ is\ the\ driver & \Rightarrow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

Rules for $\neg$

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

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

Fitch-Style Proof of Axiom A3



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

Rules for $\vee$

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

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

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 $\Leftrightarrow$

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

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

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

Proof that

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

Proof from p. 120

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

Proof from p. 119

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

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

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.	$\neg\neg Betty\ is\ the\ passenger$	$\neg I, 6\text{–}11$
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

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

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 $\ldots$ and/or $[L_n]$.

Extensional:

$[\![\{L_1, \ldots, L_n\}]\!]$ is True
if at least one of $[\![L_1]\!]$, $\ldots$, $[\![L_n]\!]$ 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

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

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

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

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

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

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:

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

To:

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

Use of Translation

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

iff

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

$\vdash \{\}$

prover Example

To prove

$(\mathit{LivingThing} \Leftrightarrow \mathit{Animal} \lor \mathit{Vegetable}), (\mathit{LivingThing} \land \neg \mathit{Animal}) \models \mathit{Vegetable}$

```
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**