Propositional Deduction

Robert Keller
15 November 200

Logic
- In CS 60 we had an introduction to both proposition- and predicate-logic.
- These were covered from the viewpoint of meaning 
  (known as “model theory” to logicians).
- There is another part of the story dealing with the 
  structure of proofs (known as “proof theory”).
- We focus on the latter now, and will connect the two 
  eventually.

Logic in CS 81
- We have two objectives in studying formal logic:
  - To firm up our concept of what forms a proof 
    and how to create proofs.
  - To investigate the connection between computability and provability, such as:
    - The problem of giving an algorithm that will determine whether or not certain kinds of statements can be proved from certain axioms is unsolvable.

Formal Systems
- A system of logical proof is a variety of formal system, just as grammars and Turing machines are formal systems.
- A formal system tells how to construct things, using precise rules, usually as some form of induction.
- “Formal” means that adherence to the rules can be checked algorithmically.

Gottlob Frege (1848-1925)
- Created modern logic by introducing the predicate calculus.
- Developed a formalized definition of “proof”.
- Defined the natural numbers in anticipation of Peano’s axiomatization (1889)
- Did not anticipate Russell’s paradox.

Varieties of Logical Proof Systems
- Axiomatic or Hilbert/Ackermann:
  - Basis is a set of axioms
  - Rules of inference tell how to derive theorems from axioms (in zero or more steps).
  - Relatively few rules of inference
- Natural Deduction, or Gentzen:
  - No axioms
  - Rules of inference tell how to derive sequents, which can entail axioms as pre-conditions and theorems as post-conditions.
  - Relatively many rules of inference.
- The two are equivalent; it is a matter of style.
Hilbert/Ackerman and Gentzen

- David Hilbert (1862-1943)
- Wilhelm Ackermann (1896-1962)
  student of Hilbert (no photo available)
- Gerhard Gentzen (1909-1945)

Natural Deduction

- A natural deduction system derives sequents, expressions of the form:
  \[ \alpha, \beta, \ldots, \gamma \]
- Each of the \( \gamma \) and \( \alpha \) represents a logical formula in an appropriate language (in the sense we have been using the term).
- The interpretation of the sequent is that each \( \gamma \) is a premise and \( \alpha \) is the conclusion.
- The \( \gamma \) could be axioms, then \( \alpha \) would be a theorem. However, the word "theorem" is usually reserved for the case that the set of premises is empty.

Truth vs. Derivation

- The intended interpretation of the sequent is that \( \alpha \) is a true formula provided that each of the \( \gamma \) are true.
- Whether or not this is really the case will depend on the rules.
- The definition of "truth" will be given later, but you can assume that it is like the one you know.
- Derivations themselves do not rely on notions of truth; they are totally mechanical.

Reference

- There are several approaches using sequents and different languages for formulas.
- We will be following the one in Huth & Ryan (HR).

A Typical Propositional Language

- E is the start symbol
- \( \text{E} \bot \text{A} \) // Atom
  \( \text{E} \bot \text{E} \) // Negation (not)
  \( \text{E} \bot \text{E} \) // Conjunction (and)
  \( \text{E} \bot \text{E} \) // Disjunction (or)
  \( \text{E} \bot \text{E} \) // Implication (implies)
  \( \text{E} \bot \) // Bottom
  \( T \) // Top
- \( \text{E} \bot 'p' \ bot 'q' \ bot 'r' \ bot 's' \ bot ... // Propositions

Bottom and Top?

- Think of bottom (\( \bot \)) as representing the constant "false".
- Think of top (\( T \)) as representing the constant "true".
Precedence

- The language as given fully parenthesizes everything.
- We will allow precedence in lieu of parentheses as an abbreviation. The binding order is negation, conjunction, disjunction, implication.

So

\[ ((p \land (q)) \land (r) \land (s \land q)) \]

could be abbreviated:

\[ (p \land q) \land (r) \land (s \land q)) \]

Examples of Sequents

- \( p, (p \land q) \rightarrow q \)
- \( (p \land q), (p \rightarrow q) \)
- \( (p \land q), (p \land r), (r \land q) \)

The first, for example, is interpreted “If \( p \) is true and \( (p \land q) \) is true, then \( q \) is true”.

More Notes on Sequents

- On the left-hand side of \( |\rightarrow \) in

\[ \Box_1, \Box_2, \ldots, \Box_n |\rightarrow \]

the formulas are regarded as a set:

- order doesn’t matter
- repetition doesn’t matter

- Order and repetition does matter within a formula. Formulas are just strings.

Sequents and Intuition

- You might be thinking “Why bother with sequents; I can do all of this with my knowledge of tautologies, etc.”
- Your knowledge can be used as intuition for validating a sequent.
- However, sequents are supposed to express whether certain deductions are valid, as they might occur in a mathematical proof.
- Tautologies won’t be enough when we introduce predicates and quantifiers.
- In addition to using sequents, we intend to study the proof systems themselves (called meta-logic).

Sequent Meta-Logical Issues

- **Soundness:**
  - Determine whether a sequent derives only true formulas from true formulas.

- **Completeness:**
  - Determine whether every true formula can be derived from a fixed set of formulas (axioms).

Natural Deduction Rules

- Each rule represents an allowable step in deriving a sequent.
- The rules focus on deriving formulas by introducing or eliminating the various connectives:

  \[ \rightarrow \]

- There is one rule for each case (introduction and elimination) for at least each connective, i.e. at least 8 rules. Some rules have multiple sub-rules.
Why “Natural” Deduction?

• “Natural” is a slogan intending to suggest that these rules are ones that might be used in normal proof construction and argumentation.

• Natural deduction also allows an argument to be developed by examining the desired conclusion and working toward assumed premises in a “natural” way.

[]-Introduction Rule ([[]])

• The reading of this rule is:
  - If [] and [] are any formulas that follow from the premises of a sequent, then the formula [] also follows from those premises.
  - The formulas above the line are called the antecedents and the one below the consequent.

Rule vs. Sequent

• Every rule immediately creates an infinite number of sequents. For example, the rule

  \[ \lambda \lambda \frac{\lambda}{\lambda} \]

  creates sequents of the form

  \[ \lambda, \lambda \frac{\lambda}{\lambda} \]

  for every pair of formulas [] and [].

  - The greek letters in the sequent form shown are not the formulas; they stand for arbitrary formulas.
  - Many sequents require multiple rule applications to establish.

Examples of Sequents Derived Using Only the ([[]]) Rule

• \( p, (q \triangleright r) \frac{\lambda}{\lambda} p[(q \triangleright r)] \) [One rule app.]

• \( p, (q \triangleright r) \frac{\lambda}{\lambda} (q \triangleright r)p \) [One rule app.]

• \( p, (q \triangleright r), s \frac{\lambda}{\lambda} ((q \triangleright r)[(p \triangleright s)]) \) [Two rule apps.]

Showing Sequent Derivations by Steps

• Derive \( p, (q \triangleright r), s \frac{\lambda}{\lambda} ((q \triangleright r)[(p \triangleright s)]) \):

  1. \( p \) Premise
  2. \( (q \triangleright r) \) Premise
  3. \( s \) Premise
  4. \( (p \triangleright s) \) Rule [[]] applied to formulas 1, 3
  5. \( ((q \triangleright r)[(p \triangleright s)]) \) Rule [[]] applied to formulas 2, 4

  - The numbers on the right refer to the antecedents used in the rule to obtain the formula on the left, which is the consequent of a rule.

Showing Sequent Derivations by DAGs

• DAG = “Directed Acyclic Graph”

  - The premises are at the leaves of the DAG.

  \[
  p \frac{\lambda}{\lambda} s \frac{\lambda}{\lambda}
  \]

  \[
  (p \triangleright s) \frac{\lambda}{\lambda} (q \triangleright r) \frac{\lambda}{\lambda}
  \]

  \[
  ((q \triangleright r)[(p \triangleright s)]) \frac{\lambda}{\lambda}
  \]

  - Note that \( (p \triangleright s) \) is used as the consequent of one rule application and the antecedent of another.
Steps vs. DAGs

- Steps correspond to the way that an argument might be presented in a math text or paper.
- DAGs allow for better visualization of what is used for what.
- Either representation can be constructed from the other.

A Step Derivation Using \(\land\) e and \(\lor\) i

- Derive \(p \land (q \lor r) : (p \land q) \lor r:\)
  1. \(p \land (q \lor r)\) Premise
  2. \(p \quad \land e_1 \quad \lor e_2 \quad 1\)
  3. \(q \lor r \quad \land e_1 \quad 1\)
  4. \(q \quad \lor e_2 \quad 3\)
  5. \(r \quad \lor e_1 \quad 3\)
  6. \(p \land q \quad \lor e_1 \quad 2, 4\)
  7. \((p \land q) \lor r \quad \lor e_2 \quad 6, 5\)

This shows that the DAG is not generally a "tree", as some antecedents are used multiple times.
Constructing Proofs by Working Backward

- If the conclusion is a premise, there is nothing to do.
- Otherwise, the outermost logical connective may suggest what rule could be used:
  - Derive \( p \land (q \lor r) \lor (p \land q) \lor r \)
  - The outermost connective in the conclusion is \( \lor \) therefore use \( \lor \) as the last step:
    - \( (p \land q) \lor r \)
    - The use of \( \lor \) will require derivation of two new formulas:
      - \( (p \land q) \lor r \)
      - Apply this approach recursively.

Choices

- Often the rule choice is not unique.
- Make a choice, but be prepared to backtrack (crossing off what you have done) and try a different one.

Constructing Proofs by Working Forward

- If a premise is the conclusion, there is nothing to do.
- Otherwise, synthesize a formula from existing formulas using available rules.
- Working forward might entail many choices of a formula to be synthesized, not all of which will be useable in deriving the conclusion.

Constructing Proofs by Working Both Directions Simultaneously

- Blend together working backward with working forward until the two "meet in the middle".
- Don’t overlook the DAG model as a means of arriving at proofs.
- Consider converting the DAG to steps for final clarity.

\(-\text{Introduction Rule ( } i_1, i_2 \)\)

\[
\begin{array}{c}
\vdash i_1 \\
\hline
\vdash i_2 \\
\hline
\vdash (i_1, i_2)
\end{array}
\]

\(-\text{Elimination Rule, Modus Ponens}\)

\[
\begin{array}{c}
\vdash \vdash (i_1, i_2) \\
\hline
\vdash \vdash (i_1) \\
\hline
\vdash \vdash (i_2)
\end{array}
\]

- Its latin name \textit{modus ponens} (MP) is often used for this rule.
Example using $\land$ -Elimination Rule

- Derive $p$, $(p \land q)$, $(q \land r) \Rightarrow r$
- $p$ Premise
- $p \land q$ Premise
- $q \land r$ Premise
- $q \Rightarrow e$ e 1, 2
- $r \Rightarrow e$ e 4, 3

- With this example, you can start to see how deriving a sequent might actually be easier (and more "natural") than establishing a tautology.

Another form of $\land$ -Elimination Rule, Modus Tollens

- A related macro or "derived rule" is modus tollens (MT):
  - $\land \Rightarrow e$ e 1, 2
  - $\Rightarrow e$ e 4, 3
  - "macro" means that this rule is a convenience and can be treated as an abbreviation for the application of other rules.
  - We will elaborate on this later.

Example using MT

- Derive $\Rightarrow r$, $(p \land q)$, $(q \land r) \Rightarrow \Rightarrow p$
- $\Rightarrow r$ Premise
- $p \land q$ Premise
- $q \land r$ Premise
- $q \Rightarrow MT$ 3, 1
- $p \Rightarrow MT$ 2, 4

$\land$ -Elimination and Introduction Rules

- $\land \Rightarrow e$ e 1, 2
- $\Rightarrow e$ e 4, 3
- (This rule is "derived").

Rules with Sub-Derivations

- Certain rules have sub-derivations, rather than simply formulas, in their antecedents.
- A sub-derivation may incorporate assumptions that behave as premises but are not premises of the sequent being proved.
- These assumptions must be treated carefully to avoid confusion with regular premises.
- Accordingly, sub-derivations are shown inside a box.
- Assumptions introduced inside the box cannot be used as premises outside the box.
- However, sub-derivations may use formulas derived earlier outside the box.

$\Rightarrow$ -Introduction Rule

- This is an example of a rule using a sub-derivation.

- Here to derive we use $\Rightarrow$ as an assumption and get $\Rightarrow$ as a conclusion using a sub-derivation.
- The sub-derivation is in a box because $\Rightarrow$ is not useable outside.
Example Using Sub-Derivation

- Derive \((p \rightarrow q), (q \rightarrow r) \mid (p \rightarrow r)\)

1. \(p \rightarrow q\) \hspace{1cm} \text{Premise}
2. \(q \rightarrow r\) \hspace{1cm} \text{Premise}
3. \(p\) \hspace{1cm} \text{Assumption}
4. \(q\) \hspace{1cm} \text{\(\square\) e 1, 2}
5. \(r\) \hspace{1cm} \text{\(\square\) e 2, 4}
6. \(p \rightarrow r\) \hspace{1cm} \text{\(\square\) i 2-5}

Another Example Using Sub-Derivation

- Derive \((\square p \rightarrow q) \mid (\square q \rightarrow p)\)

1. \(\square p \rightarrow q\) \hspace{1cm} \text{Premise}
2. \(q\) \hspace{1cm} \text{Assumption}
3. \(\square q\) \hspace{1cm} \text{\(\square\) i 2}
4. \(\square p\) \hspace{1cm} \text{MT 1, 3}
5. \(p\) \hspace{1cm} \text{\(\square\) e 4}
6. \(q \rightarrow p\) \hspace{1cm} \text{\(\square\) i 2-5}

- Pattern matching:
  - \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\)
  - \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\)
  - \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\)
  - \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\) \(\square\)

A Sub-Derivation can be Trivial

- Derive \((\square p \rightarrow p)\) (Set of premises is empty):

1. \(p \rightarrow p\) \hspace{1cm} \text{Assumption}
2. \(p \rightarrow p\) \hspace{1cm} \text{\(\square\) i 1, 1}

- Pattern matching:
  - \(\square\)
  - \(\square\)
  - \(\square\)
  - \(\square\)
  - \(\square\)
  - \(\square\)

Both \(\square\) and \(\square\) are \(p\).

Sub-Derivations can be Nested

- Derive \((p \rightarrow q) \rightarrow p \rightarrow (q \rightarrow r)\)

1. \(p \rightarrow q\) \hspace{1cm} \text{Premise}
2. \(p\) \hspace{1cm} \text{Assumption}
3. \(q\) \hspace{1cm} \text{\(\square\) i 2, 3}
4. \(p \rightarrow q\) \hspace{1cm} \text{\(\square\) e 1, 4}
5. \(r\) \hspace{1cm} \text{\(\square\) i 1-3, 5}
6. \(q \rightarrow r\) \hspace{1cm} \text{\(\square\) i 2-6}
7. \(p \rightarrow (q \rightarrow r)\) \hspace{1cm} \text{\(\square\) i 2-6}

Sub-Derivations and DAGs

- It is unclear how to show sub-derivations in the DAG model.
- The customary way is to introduce the sub-derivation and discharge (cross-out) the assumptions so that they cannot be used outside the sub-derivation.
- The steps model is clearer in this regard, because nesting shows the order of discharge.

Sub-Derivations in the DAG model

- Derive \((p \rightarrow q) \rightarrow r \rightarrow p \rightarrow (q \rightarrow r)\)

<table>
<thead>
<tr>
<th>Assumption (/ denotes discharged)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
<tr>
<td>(\square) (\square) (\square) (\square) (\square) (\square) (\square)</td>
</tr>
</tbody>
</table>
-Elimination Rule

- This rule uses two sub-derivations:

- The interpretation is that if we want to "get rid of" a disjunction, we can derive a common formula from the two disjuncts.

Sub-Derivations vs. Sequents?

- Aren't the boxed sub-derivations essentially sequents themselves?
- If so, why don't we use the notation \[ \frac{}{e} \] rather than

- The answer probably lies in the fact that sub-derivations can make use of formulas outside the box, and we'd have to repeat those formulas as premises of the sequent.

-Introduction Rule

- This rule introduces \[ i \] through "contradiction":

-Elaboration Rule

- If we can derive \[ i \] then we can derive anything. Consequently, the things we derive won't have much information value. So being able to derive \[ i \] is undesirable, except in a sub-derivation.

Macro or Derived Rules

- Earlier MT was mentioned as a "macro" rule.
- The name "macro" alludes to programming language macros.
- While superficially similar to a subroutine, a macro is a text substitution done before a source is compiled or interpreted.
- In our case, it is a rule that could be replaced with a sequence of uses of other rules.
MT as a Macro derived from other rules

\[
\begin{array}{cc}
1. & \text{Premise} \\
2. & \text{Premise} \\
3. & \text{Assumption} \\
4. & \text{Premise} \\
5. & \text{Premise} \\
6. & \text{Assumption} \\
\end{array}
\]

• Every use of MT could thus be replaced with this sequence, which uses 3 rules: \(\neg, \wedge, \neg\).

\[
\begin{array}{c}
\text{\(\neg\neg\neg\neg\neg\neg\neg\)}
\end{array}
\]

Macro vs. Sequent

• Why isn’t a macro rule just another sequent?

RAA (Reductio ad absurdum) Rule

• This rule has a similarity to \(\neg\neg\):

\[
\begin{array}{c}
\text{\(\neg\neg\neg\neg\neg\neg\neg\)}
\end{array}
\]

RAA as a Macro derived from other rules

\[
\begin{array}{c}
\text{\(\neg\neg\neg\neg\neg\neg\neg\)}
\end{array}
\]

LEM (Law of the Excluded Middle)

• \(\neg\neg\) (No antecedent)

\[
\begin{array}{c}
\text{\(\neg\neg\neg\neg\neg\neg\neg\)}
\end{array}
\]
Summary of Non-Derived Rules

<table>
<thead>
<tr>
<th>Connective</th>
<th>Introduction</th>
<th>Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>⊢ e₁, e₂</td>
</tr>
<tr>
<td></td>
<td>l₁, l₂</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>i₁, i₂</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>(none)</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>(derived)</td>
<td>⊢ e</td>
</tr>
</tbody>
</table>

Summary of Derived Rules So Far

- MT (Modus Tollens)
- RAA (Reductio ad Absurdum)
- LEM (Law of the Excluded Middle)
- ⊢ [n]

Validity vs. Provability

- $\mathcal{G}_1, ..., \mathcal{G}_n \vdash \mathcal{G}$ means $\mathcal{G}$ is provable from $\mathcal{G}_1, ..., \mathcal{G}_n$.
- $\mathcal{G}_1, ..., \mathcal{G}_n \models \mathcal{G}$ means roughly the following:
  - If each of $\mathcal{G}_1$ is true, then $\mathcal{G}$ is true.
- In other words, $\mathcal{G}$ is a valid conclusion from $\mathcal{G}_1, ..., \mathcal{G}_n$.
- We need a definition of truth to make this precise.