CS 3110

Lecture 19: Logic
To Truth through Proof

Prof. Clarkson
Fall 2014

Today’s music: Theme from Sherlock
Review

Current topic:
- How to reason about correctness of code
- Last week: informal arguments

Today:
- Logic
- Necessary step on our way to having machine-checked proofs of correctness
Question #1

What is your background in logic?
A. I've never studied any formal logic AFAIK.
B. I saw a little bit in CS 2800.
C. I've taken a CS logic class.
D. I've taken a math logic class.
E. I've taken a philosophy logic class.
A biased history of logic

• Originated in philosophy
• Mathematicians became interested in late 1800s and early 1900s
  – goal: formalize mathematical reasoning
  – impossible: Kurt Gödel
• Computer scientists found many applications in the late 20th century
  – AI: formalize reasoning of robots, agents
  – Theorem proving: verify mathematical theorems, even discover new theorems
  – Verification: prove correctness of programs!
A biased perspective on logic

• A logic is a programming language for expressing reasoning about evidence
  – Like how OCaml is a programming language for expressing computation on data (ints, bools, etc.)
  – Data and evidence are analogous
  – Computation and reasoning are analogous

• Like any PL, a logic has
  – syntax
  – dynamic semantics (evaluation rules) --omitted here
  – static semantics (type checking)
A logic: IPC

Syntax:
\[ f ::= P \mid f_1 \land f_2 \mid f_1 \lor f_2 \mid f_1 \implies f_2 \mid \neg f \]

• \( f \) is a meta-variable for formulae
• \( P \) is a meta-variable for propositions
  – We'll use any capital letter for propositions
  – except: \textbf{true} and \textbf{false} are also propositions
A logic: IPC

Syntax:
\[ f ::= P \mid f_1 \land f_2 \mid f_1 \lor f_2 \mid f_1 \Rightarrow f_2 \mid \neg f \]

- \( \land \) is logical and (aka conjunction)
- \( \lor \) is logical or (aka disjunction)
- \( \Rightarrow \) is logical implication
- \( \neg \) is logical negation
  - actually syntactic sugar: \( \neg f \) means \( f \Rightarrow \) false
A logic: IPC

Syntax:

\[ f ::= P \mid f_1 \land f_2 \mid f_1 \lor f_2 \mid f_1 \rightarrow f_2 \mid \neg f \]

• Note on notation:
  – Slides use an ASCII syntax
  – Online notes use nicer math symbols
  – Either is fine, but be consistent

• IPC= Intuitionistic Propositional Calculus
**Formal syntax**

- Abstracts from ambiguities and details of natural language
- Examples:
  - *Mammals have hair. Monkeys have hair. So monkeys are mammals.*
  - *Mammals have hair. Teddybears have hair. So teddybears are mammals.*
  - \((M \implies H) \land (X \implies H)\) \implies (X \implies M)
  - All are flawed reasoning!
    - (Want a way to distinguish flawed reasoning from correct reasoning...)
- A logic is a precise way to express structure of reasoning
- Just like a PL is a precise way to express structure of computation
Parts of syntax

• Connectives
  – and \( \land \), or \( \lor \), implies \( \implies \), not \( \sim \)
  – like binary operators in a PL
  – create larger formulae (expressions) out of smaller

• Propositions
  – the basic "atoms" being reasoned about
  – like built-in data types (int, bool) in a PL
  – the simplest kind of formulae (expressions)
Formalization of an argument

• If there is a snowstorm, then the roads will be closed.
• The roads are open.
• So there can't be a snowstorm.
Formalization of an argument

• If there is a snowstorm, then the roads will be closed. \( S \implies C \)
• The roads are open.
• So there can't be a snowstorm.
Formalization of an argument

• If there is a snowstorm, then the roads will be closed. \( S \Rightarrow C \)
• The roads are open. \( O \)
• So there can’t be a snowstorm.
Formalization of an argument

• If there is a snowstorm, then the roads will be closed.  \( S \rightarrow C \)
• The roads are open.  \( O \)
• So there can't be a snowstorm.  \( \sim S \)
Formalization of an argument

• If there is a snowstorm, then the roads will be closed.  \( S \rightarrow C \)
• The roads are open.  \( O \)
• So there can’t be a snowstorm.  \( \sim S \)
• Implicit: A road is either open or closed.  
  \( O \rightarrow \sim C \)  \( \lor \)  \( C \rightarrow \sim O \)
Formalization of an argument

• If there is a snowstorm, then the roads will be closed.  \( S \Rightarrow C \)
• The roads are open.  \( O \)
• So there can't be a snowstorm.  \( \sim S \)
• *Implicit*: A road is either open or closed.
  \( O \Rightarrow \sim C \quad \land \quad C \Rightarrow \sim O \)
• Combining them all:
  \[
  ((S \Rightarrow C) \land O \land ((O \Rightarrow \sim C) \land (C \Rightarrow \sim O))) \Rightarrow \sim S
  \]
Question #2

Which subformula does not appear in formalization?

If there is a snowstorm then the roads will be closed. There is no snowstorm. So the roads must be open.

A. S=>C
B. ~S
C. C=>S
D. O
E. O=>~C
Question #2

Which subformula does not appear in formalization?

If there is a snowstorm then the roads will be closed. There is no snowstorm. So the roads must be open.

A. S=>C  
B. ~S  
C. C=>S  
D. O  
E. O=>~C
Valid vs. invalid arguments

• How to separate them?
• What constitutes correct reasoning?
• Analogy: how did we distinguish "valid" from "invalid" programs?
  – Static semantics = type system
• So let's build a "type system" for valid arguments
  – Usually called a "proof system" or "deductive system"
Proof system for IPC

• Only one type: **provable**
  – e.g., \((A \land B) \Rightarrow A : \text{provable}\)
  – e.g., \(A \Rightarrow (A \land B)\) is not provable so can't be given a type

• No reason to keep writing "\(f : \text{provable}\)" everywhere
  – the colon and word "provable" are too verbose

• Instead, write \(\vdash f\)
  – pronounced as "provable \(f\)" or "\(f\) is provable"
  – or "derivable" instead of "provable"
Proof system for IPC

• We'll give *proof rules* for each syntactic form in IPC

• Just like we gave *type-checking rules* for each syntactic form in OCaml

  \[
  5 : \text{int} \\
  \text{fun } x \to e : \text{ta} \to \text{tb} \text{ if } e : \text{tb} \text{ under assumption } x : \text{ta}
  \]
Proof system for IPC

• We'll give introduction and elimination rules for each form
• Just like we gave rules for building and accessing pieces of data in OCaml
  – \((e_1, e_2) : a*b \text{ if } e_1 : a \text{ and } e_2 : b\)
  – \(\text{fst } e : a \text{ if } e : a*b\)

All rules will be based on evidence for each form...
Evidence for $\land$

Q: What constitutes evidence for $f_1 \land f_2$?

A: Evidence for both $f_1$ and $f_2$, individually

so evidence for $f_1 \land f_2$ is really a pair of the evidence for $f_1$ and the evidence for $f_2$...
Proof rules for $\land$

• if $\vdash f_1$ and $\vdash f_2$ then $\vdash f_1 \land f_2$

  – **introduction rule:** shows how to build/introduce a formula out of smaller pieces

  – **intuition:** if you have evidence for $f_1$ and evidence for $f_2$, then you can combine those pieces of evidence to get evidence for $f_1 \land f_2$
Proof rules for $\land$

- if $\vdash f_1 \land f_2$ then $\vdash f_1$
- if $\vdash f_1 \land f_2$ then $\vdash f_2$

- **elimination rules:** show how to access smaller formulae out of larger, i.e., eliminate parts of formulae

- **intuition:** if you have evidence for $f_1 \land f_2$, then you can break apart that to get evidence for $f_1$ individually, likewise for $f_2$

- **further intuition:** these rules are really just $\text{fst}$ and $\text{snd}$
Evidence for =>

Q: What constitutes evidence for $f_1 \Rightarrow f_2$?
A: A way to transform evidence for $f_1$ into evidence for $f_2$.

So evidence for $f_1 \Rightarrow f_2$ is really a function that transforms evidence for $f_1$ into evidence for $f_2$...
Proof rules for =>

• if \( \vdash \textit{f} \) and \( \vdash \textit{f} \Rightarrow \textit{g} \) then \( \vdash \textit{g} \)

  – traditionally called \textit{modus ponens}: "way that affirms"

  – elimination rule

  – \textbf{intuition}: if you have evidence for \( \textit{f} \), and you have a way of transforming evidence for \( \textit{f} \) into evidence for \( \textit{g} \), then you have evidence for \( \textit{g} \)

  – \textbf{further intuition}: this rule is really just function application
Proof rules for $\Rightarrow$

- if under the assumption $\vdash f$ we can conclude $\vdash g$, then $\vdash f \Rightarrow g$
  - introduction rule
  - **intuition:** the way you reached that conclusion must be a way of transforming evidence for $f$ into evidence for $g$, so you have evidence for $f \Rightarrow g$
  - further intuition: this rule is really just anonymous function definition
  - **hypothetical reasoning:** "if I assume $X$, then I can conclude $Y"."
Notation for assumptions

- $f \vdash g$ means "under the assumption that $f$ is provable, it holds that $g$ is provable"

- So instead of:
  
  if under the assumption $\vdash f$ we can conclude $\vdash g$, then $\vdash f \Rightarrow g$

  we can write:

  if $f \vdash g$ then $\vdash f \Rightarrow g$

- Generalize to entire set of assumptions: $F \vdash g$ means "under the assumption that all formulas in set $F$ are provable, it holds that $g$ is provable"
  
  - Write comma instead of set union: $F, f$ means $F \cup \{f\}$
Revised proof rules

Adding assumptions to all rules so far:

• if $F |- f_1$ and $F |- f_2$ then $F |- f_1 \land f_2$

• if $F |- f_1 \land f_2$ then $F |- f_1$

• if $F |- f_1 \land f_2$ then $F |- f_2$

• if $F |- f$ and $F |- f \implies g$ then $F |- g$

• if $F, f |- g$ then $F |- f \implies g$
Proof rules for assumptions

• \( f \vdash f \)
  – Intuition: if you have assumed that you have evidence for \( f \), then you can proceed as though you have evidence for \( f \)
  – This rule is an *axiom*: it has no premises

• if \( F \vdash f \) then \( F, g \vdash f \)
  – Intuition: if assuming \( F \) is enough to derive evidence for \( f \), then additionally assuming \( g \) makes no difference
  – This rule is called *weakening*: assuming more weakens the claim
A proof

Let's show \( \neg (A \Rightarrow (B \Rightarrow A)) \)

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \land ) intro</td>
<td>if ( F \vdash f_1 ) and ( F \vdash f_2 ) then ( F \vdash f_1 \land f_2 )</td>
</tr>
<tr>
<td>( \land ) elim L</td>
<td>if ( F \vdash f_1 \land f_2 ) then ( F \vdash f_1 )</td>
</tr>
<tr>
<td>( \land ) elim R</td>
<td>if ( F \vdash f_1 \land f_2 ) then ( F \vdash f_2 )</td>
</tr>
<tr>
<td>( \Rightarrow ) elim</td>
<td>if ( F \vdash f ) and ( F \vdash f \Rightarrow g ) then ( F \vdash g )</td>
</tr>
<tr>
<td>( \Rightarrow ) intro</td>
<td>if ( F, f \vdash g ) then ( F \vdash f \Rightarrow g )</td>
</tr>
<tr>
<td>assume</td>
<td>( f \vdash f )</td>
</tr>
<tr>
<td>weak</td>
<td>if ( F \vdash f ) then ( F, g \vdash f )</td>
</tr>
</tbody>
</table>
A proof

Let's show $\vdash (A \Rightarrow (B \Rightarrow A))$

1. $A \vdash A$ by assumption rule
2. $A, B \vdash A$ by (1) and weakening rule
3. $A \vdash B \Rightarrow A$ by (2) and $\Rightarrow$ introduction rule
4. $\vdash A \Rightarrow (B \Rightarrow A)$ by (3) and $\Rightarrow$ introduction rule
Proof structure

• Each step numbered
• Each step derives one new formula from previous step(s) and from named rule
• At each rule application, all the premises of a rule must already have been derived. Get to add conclusion of rule as new numbered step.
• Final step is the formula we want to prove, with no assumptions
A graphical notation: proof trees

\[
\begin{align*}
A & \quad |\quad - \quad A \\
A, B & \quad |\quad - \quad A \\
& \quad \quad \Rightarrow \text{intro.} \\
A & \quad |\quad - \quad B \Rightarrow A \\
& \quad \quad \Rightarrow \text{intro.} \\
\Rightarrow & \quad (A \Rightarrow (B \Rightarrow A))
\end{align*}
\]
Proof structure

• Goal formula is at root of tree (bottom)
• Each node in tree is a formula
  – i.e., a numbered step from linear form
• Each edge in tree is labeled by rule name
  – i.e., a justification from linear form
• If rule has no premises, there's an "empty" node at top
  – i.e., an axiom
That proof as an OCaml program

```ocaml
let t (a:'a) (b:'b) : 'a = a
```

How to think about this program:
- `t` is a function that takes in evidence for `a`, evidence for `b`, and returns the evidence for `a`

What is its type?
```ocaml
'a -> ('b -> 'a)
```

What is the formula we proved?
```
A => (B => A)
```
Programs and Proofs

• We were able to write a program whose type is the very formula we were trying to prove

• That program is an evidence transformer: it manipulates input evidence to construct output evidence of the right type

• This correspondence between
  – programs and proofs
  – types and formulae
  goes very, very deep.

• Known as the Curry-Howard isomorphism
Another proof

Let's show $\vdash A \Rightarrow (B \Rightarrow (A \land \neg B))$.

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\land$ intro</td>
<td>if $F \vdash f_1$ and $F \vdash f_2$ then $F \vdash f_1 \land f_2$</td>
</tr>
<tr>
<td>$\land$ elim L</td>
<td>if $F \vdash f_1 \land f_2$ then $F \vdash f_1$</td>
</tr>
<tr>
<td>$\land$ elim R</td>
<td>if $F \vdash f_1 \land f_2$ then $F \vdash f_2$</td>
</tr>
<tr>
<td>=&gt; elim</td>
<td>if $F \vdash f$ and $F \vdash f \Rightarrow g$ then $F \vdash g$</td>
</tr>
<tr>
<td>=&gt; intro</td>
<td>if $F, f \vdash g$ then $F \vdash f \Rightarrow g$</td>
</tr>
<tr>
<td>assms</td>
<td>$f \vdash f$</td>
</tr>
<tr>
<td>weak</td>
<td>if $F \vdash f$ then $F, g \vdash f$</td>
</tr>
</tbody>
</table>
Another proof: linear form

Let’s show $\vdash A \Rightarrow (B \Rightarrow (A \land B))$.

1. $A \vdash A$ by assumption rule
2. $A, B \vdash A$ by weakening rule
3. $B \vdash B$ by assumption rule
4. $A, B \vdash B$ by weakening rule
5. $A, B \vdash A \land B$ by (2), (4), and $\land$ introduction rule
6. $A \vdash B \Rightarrow (A \land B)$ by (5) and $\Rightarrow$ introduction rule
7. $\vdash A \Rightarrow (B \Rightarrow (A \land B))$ by (6) and $\Rightarrow$ introduction rule
Another proof: tree form

\[
\begin{align*}
A & | - A \\
& \text{assump} \\
A & | - A \\
& \text{weak} \\
A, B & | - A \\
& \text{weak} \\
A, B & | - A \\
& \text{weak} \\
A, B & | - A \land B \\
& \text{weak} \\
A, B & | - A \land B \\
& \text{weak} \\
A & | - B \Rightarrow (A \land B) \\
& \text{weak} \\
| - A & \Rightarrow (B \Rightarrow (A \land B)) \\
& \text{weak} \\
\end{align*}
\]
As an OCaml program

```ocaml
let pair (a:'a) (b:'b) : ('a*'b)
   = (a,b)
```

How to think about this program:
`pair` is a function that takes in evidence for `a`, evidence for `b`, and returns the pair containing both pieces of evidence.

What is its type?
'\(a \rightarrow (b \rightarrow (a \times b))\)

What is the formula we proved?
A \(\Rightarrow (B \Rightarrow (A \land B))\)
Please hold still for 1 more minute

WRAP-UP FOR TODAY
Upcoming events

• PS5 checkins this week
• Clarkson office hour today cancelled; moved to tomorrow
• Thursday: Guest lecture by Yaron Minsky (Cornell PhD) from Jane Street on "OCaml in the Real World"

This is logical.