CS 3110

Lecture 21: Logic, part II

To Truth through Proof

Prof. Clarkson

Fall 2014

Today's music: "The Devil went down to Georgia"
by The Charlie Daniels Band

Review

Current topic:

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

Today:

- Logic, part II
- Upgrade from *propositional* logic to *predicate* logic

Question #1

How much of PS5 have you finished?

A. None
B. About 25%
C. About 50%
D. About 75%
E. I'm done!!!

Review: A biased perspective on logic

- A *logic* is a programming language for expressing reasoning about evidence
- Like any PL, a logic has
  - **syntax**
  - **dynamic semantics** (evaluation rules) *--*omitted here
  - **static semantics** (type checking)

Review: IPC

IPC= Intuitionistic Propositional Calculus

Syntax:

```
f ::= P | f1 /\ f2 | f1 \/ f2
    | f1 => f2 | ~f
P ::= true | false | ...
```

Review: Proof rules so far

Rule name	Rule
/\ intro	if `F |- f1` and `F |- f2` then `F |- f1 /\ f2`
/\ elim L	if `F |- f1 /\ f2` then `F |- f1`
/\ elim R	if `F |- f1 /\ f2` then `F |- f2`
=> elim	if `F |- f` and `F |- f => g` then `F |- g`
=> intro	if `F, f |- g` then `F |- f => g`
assump	`f |- f`
weak	if `F |- f` then `F,g |- f`
set assump	`F,f |- f`

Evidence for `true` and `false`

Q: What constitutes evidence for **`true`**?

A: We don't need any; **`true`** trivially holds

Q: What constitutes evidence for **`false`**?

A: Nothing; **`false`** can never hold.

If we ever did somehow have evidence for **`false`**, *then we'd be in a contradictory situation, and all reason has broken down.*

Proof rules for `true` and `false`

- `F |- true`
  - only an introduction rule, no elimination
  - another axiom
  - intuition: we can always give evidence for true
- if `F |- false` then `F |- f`
  - *ex falso quodlibet*: "from false follows whatever you please"
  - Principle of Explosion
  - only an elimination rule, no introduction
  - intuition: we can never give evidence for `false`; but once we can conclude `false`, we can conclude anything

Evidence for ~

Q: What constitutes evidence for **`~f`**?

A: Since **`~f`** really means **`f=>false`**, it would be a way of transforming evidence for **`f`** into evidence for false. That is, a way of reaching a contradiction.

Proof rules for ~

Negation is just syntactic sugar, so free to convert between those two forms:

- if `F |- f => false` then `F |- ~f`
  - intuition: if there's a way to transform evidence for `f` into evidence for `false`, then you have evidence for `~f`
- if `F |- ~f` then `F |- f => false`
  - intuition: if you have evidence for `~f`, then you have a way of transforming evidence for `f` into evidence for `false`

Evidence for \/

Q: What constitutes evidence for **`f1\/f2`**?

A: Evidence for either **`f1`** or for **`f2`**, tagged to indicate which one it's evidence for.

So evidence for **`f1\/f2`** is really a value of a **datatype**:

```
type ('a,'b) sum =
  Left of 'a | Right of 'b
```

Proof rules for \/

- if `F |- f1` then `F |- f1 \/ f2`
- if `F |- f2` then `F |- f1 \/ f2`
  - intuition: if you have evidence for **`f1`**, then you have evidence for **`f1\/f2`**
  - further intuition: these rules are really just constructor application

Proof rules for \/

- if `F |- f1 \/ f2` and `F |- f1 => g` and `F |- f2 => g` then `F |- g`
  - intuition: if you have evidence for `f1\/f2`, and if you have a way of transforming evidence for `f1` into evidence for `g`, as well as for `f2` into `g`, then you can obtain evidence for `g`
  - further intuition: this rule is really just pattern matching!

```
match s with
  Left f1 -> e1
| Right f2 -> e2
```

Proof with \/

Let's show `|- (P \/ Q) => (Q \/ P)`

1. `P \/ Q |- P \/ Q` by assump
2. `P |- P` by assump
3. `P |- Q \/ P` by (2) and \/ intro R
4. `|- P => Q \/ P` by (3) and **=>** intro
5. `P \/ Q |- P => Q \/ P` by (4) and weak.
6. `Q |- Q` by assump
7. `Q |- Q \/ P` by (6) and \/ intro L
8. `|- Q => Q \/ P` by (7) and **=>** intro
9. `P \/ Q |- Q => Q \/ P` by (8) and weak.
10. `P \/ Q |- Q \/ P` by (1), (5), (9) and \/ elim
11. `|- (P \/ Q) => (Q \/ P)` by **=>** intro

Tree form

```
|- (P \/ Q) => (Q \/ P)
```

Tree form

```
P \/ Q |- Q \/ P
```

———————————————— => intro

```
|- (P \/ Q) => (Q \/ P)
```

Tree form

```
P\/Q |- P\/Q      P\/Q |- P=>(Q\/P)      P\/Q |- Q=>(Q\/P)
------------------------------------------------------------ \/ elim
                  P \/ Q |- Q \/ P
           ---------------------------- => intro
           |- (P \/ Q) => (Q \/ P)
```

Tree form

```
                                assump
-----------------------------------
P\/Q |- P\/Q          P\/Q |- P=>(Q\/P)          P\/Q |- Q=>(Q\/P)
--------------------------------------------------------------------------------
                                                                          \/ elim
                    P \/ Q |- Q \/ P
              ----------------------------------- => intro
                 |- (P \/ Q) => (Q \/ P)
```

Tree form

```
                                P\/Q, P |- Q\/P
                   assump       ------------------------ => intro
-------------------
P\/Q |- P\/Q                    P\/Q |- P=>(Q\/P)        P\/Q |- Q=>(Q\/P)
-------------------------------------------------------------------------- \/ elim
                                P \/ Q |- Q \/ P
                            ------------------------ => intro
                            |- (P \/ Q) => (Q \/ P)
```

Tree form

```
                          P |- Q\/P
                      ------------------- weak
                      P\/Q, P |- Q\/P
            assump    ---------------------- => intro
-----------------
P\/Q |- P\/Q          P\/Q |- P=>(Q\/P)        P\/Q |- Q=>(Q\/P)
--------------------------------------------------------------------
                                                              \/ elim
                      P \/ Q |- Q \/ P
                  ---------------------------- => intro
                  |- (P \/ Q) => (Q \/ P)
```

Tree form

```
                                  ------------  assump
                                    P |- P
                                 --------------  \/ intro-r
                                   P |- Q\/P
                              ------------------------  weak
                               P\/Q, P |- Q\/P
                     assump                                   => intro
------------------------   ----------------------------
P\/Q |- P\/Q               P\/Q |- P=>(Q\/P)          P\/Q |- Q=>(Q\/P)
--------------------------------------------------------------------------------
                                                                        \/ elim
                            P \/ Q |- Q \/ P
                        ------------------------------------  => intro
                         |- (P \/ Q) => (Q \/ P)
```

Tree form

```
 ----------- assump              ----------- assump
  P |- P                          Q |- Q
 ----------- \/ intro-r           ----------- \/ intro-l
  P |- Q\/P                       Q |- Q\/P
 ------------------ weak          ------------------ weak
  P\/Q, P |- Q\/P                 P\/Q, Q |- Q\/P
              assump
 ---------------  ------------------ => intro  ------------------ => intro
 P\/Q |- P\/Q     P\/Q |- P=>(Q\/P)           P\/Q |- Q=>(Q\/P)
 ------------------------------------------------------------------ \/ elim
                  P \/ Q |- Q \/ P
 ------------------------------------ => intro
                  |- (P \/ Q) => (Q \/ P)
```

Note: bad formatting! hard to fit on slide ☹

As an OCaml program

```
let or_comm (s: ('p,'q) sum)) : ('q,'p) sum =
  match s with
    Left p -> Right p
  | Right q -> Left q
```

How to think about this program:

`or_comm` is a function that takes in evidence for either `'p` or `'q`, and returns evidence for either `'q` or `'p`

As an OCaml program

```
let or_comm (s: ('p,'q) sum) : ('q,'p) sum =
  match s with
    Left p -> Right p
  | Right q -> Left q
```

What is its type?

`('p, 'q) sum -> ('q, 'p) sum`

imagine we could write **`sum`** as infix **+**...

`'p + 'q -> 'q + 'p`

What is the formula we proved?

`(P \/ Q) => (Q \/ P)`

What about `P\/ (~P)`?

- aka *excluded middle*
- Many presentations of logic simply assume this holds for any proposition **`P`**
  - Indeed, for any formula **`f`**
- **Cannot be proved in IPC**
- But we could add **`|- P \/ (~P)`** to IPC to get a new logic, CPC
  - CPC has same **syntax** as IPC, but **type system** that's "bigger" by one rule
  - Then we'd be saying there's always a way to give evidence for either **`P`**, or for **`P=>false`**.
  - But we wouldn't be saying what that evidence is...


The Devil's Middle

Classical vs. constructive

- Without excluded middle we have *constructive logic*
  - Constructive ≅ intuitionistic
  - A *constructive* proof is an algorithm (cf. the programs we've been writing that correspond to proofs)
- With it, we have *classical logic*
  - CPC = classical propositional calculus
- Truth vs. proof
  - Truth:
    - Classical proofs are concerned with truth values
    - All propositions are either true or false
  - Proof:
    - Constructive proofs are concerned with evidence
    - Propositions don't have "truth values"; rather, their truth is unknown until can be (dis)proved

Proof rules of IPC, part 1

<table>
<tr><th>Rule name</th><th>Rule</th></tr>
<tr><td>/\ intro</td><td>if F |- f1 and F |- f2 then F |- f1 /\ f2</td></tr>
<tr><td>/\ elim L</td><td>if F |- f1 /\ f2 then F |- f1</td></tr>
<tr><td>/\ elim R</td><td>if F |- f1 /\ f2 then F |- f2</td></tr>
<tr><td>=> elim</td><td>if F |- f and F |- f => g then F |- g</td></tr>
<tr><td>=> intro</td><td>if F, f |- g then F |- f => g</td></tr>
<tr><td>assump</td><td>f |- f</td></tr>
<tr><td>weak</td><td>if F |- f then F,g |- f</td></tr>
<tr><td>set assump</td><td>F,f |- f</td></tr>
</table>

Proof rules of IPC, part 2

Rule name	Rule
\/ intro L	if `F |- f1` then `F |- f1 \/ f2`
\/ intro R	if `F |- f2` then `F |- f1 \/ f2`
\/ elim	if `F |- f1 \/ f2` and `F |- f1 => g` and `F |- f2 => g` then `F |- g`
true intro	`F |- true`
false elim	if `F |- false` then `F |- f`
~ intro	if `F |- f => false` then `F |- ~f`
~ elim	if `F |- ~f` then `F |- f => false`

Natural deduction

- Style of proof system we just gave is called *natural deduction*
  - Gentzen (1934), Prawitz (1965)
  - Very few axioms, mostly *inference rules*
  - With intro and elim rules for each connective
- Graphical notation for proof trees is considered a strength of this style
  - Even if it doesn't work well in slides! ☺
  - Even if it doesn't scale well to large proofs!
- In notes and in recitation: larger examples of proofs

Formalize this argument

- All squares are positive
- 9 is a square
- Therefore 9 is positive

Formalize this argument

- All squares are positive `f`
- 9 is a square `g`
- Therefore 9 is positive `h`

an attempt: `f /\ g => h`

...but that's not a provable formula

...so we might have trouble proving that
the return value of **`square`** is positive!

...we need *predicates*

Predicates

- *Predicates* aka *relations* upgrade propositions to have arguments:
  - `is_positive(x)`
  - `is_square(x)`
  - `equals(x,y)`, usually written `x=y`
- *Objects* (the variables above) are the atomic things we now talk about
  - might be integers, lists of strings, real numbers, etc.
- *Functions* map between objects
  - `square(3)`, which is `9`
- *Quantifiers* let us talk about all objects at once:
  - "for all objects x, it holds that `P(x)`" (universal)
  - "there exists an object x, such that `P(x)` holds" (existential)

A new logic: IQC

Syntax:

```
f ::= P(t1,...,tn)
    | f1 /\ f2 | f1 \/ f2
    | f1 => f2 | ~f
    | forall x, f
    | exists x, f
 t ::= x | fn(t1,...,tn)
```

- `P` is a meta-variable for predicates/relations (incl. *nullary* predicates `true` and `false`)
- `t` is a meta-variable for *terms*, including constants, variables, and functions `fn` applied to terms (including *nullary* functions, i.e., constants)

IQC

- IQC = Intuitionistic Quantifier Calculus
- CQC = Classical Quantifier Calculus
  - equals IQC + excluded middle
- CQC aka
  - first order logic (FOL)
  - predicate logic
  - predicate calculus

Formalize this argument

- All squares are positive `forall x, is_square(x) => is_positive(x)`
- 9 is a square `is_square(9)`
- Therefore 9 is positive `is_positive(9)`

```
((forall x, is_square(x) => is_positive(x))
 /\ is_square(9))
=> is_positive(9)
```

Proof rules for IQC

- All the rules of IPC, plus intro and elim for quantifiers
- New notation:
  - **`f(x)`** means a formula **`f`** that mentions a variable **`x`**
  - **`f(t)`** means that same formula **`f`**, but with all mentions of **`x`** replaced by term **`t`**

Evidence for `forall`

Q: What constitutes evidence for `forall x, f(x)`?

A: A way of producing evidence for `f(x)` out of an arbitrary object `x`.

...That is, a way of transforming an object `x` into evidence of `f(x)`

(note the similarity to `=>`)

Proof rules for `forall`

- if `F |- f(x)` and **`F`** does not make any assumptions about **`x`**, then `F |- forall x, f(x)`
  - introduction rule
  - **intuition:** if you can construct evidence for **`f(x)`** without making any assumptions about **`x`**, then you have a way of transforming **`x`** into evidence for **`f(x)`**

...but what does "make assumptions about" mean"?

Free variables

Free variables are variables that aren't bound by any quantifer

- **`P(x)`**: **`x`** is free
- **`forall x, P(x) /\ Q(y)`**: **`x`** is not free and **`y`** is free
- **`R(x) => (forall x, P(x))`**: **`x`** is free in LHS of implication, but not in RHS

If **`x`** does not occur free in a formula, then the formula makes no assumptions about it.

Likewise for a set of formulae.

Free variables (formal defn)

```
FV(x) = {x}
FV(f(t1,...tn)) = FV(t1) ∪... ∪ FV(tn)
FV(P(t1,...tn)) = FV(t1) ∪... ∪ FV(tn)
FV(f1/\f2) = FV(f1) ∪ FV(f2)
FV(f1=>f2) = FV(f1) ∪ FV(f2)
FV(f1\/f2) = FV(f1) ∪ FV(f2)
FV(~f) = FV(f)
FV(forall x, f) = FV(f) \ {x}
FV(exists x, f) = FV(f) \ {x}
```

Proof rules for `forall`

- if `F |- f(x)` and **`x`** does not *occur free* in **`F`**, then `F |- forall x, f(x)`
  - introduction rule
  - "**`x`** does not *occur free* in **`F`**" means **`x`** not in **`FV(f)`** for any **`f`** in **`F`**
  - **intuition:** if you can construct evidence for **`f(x)`** without making any assumptions about **`x`**, then you have a way of transforming **`x`** into evidence for **`f(x)`**

Proof rules for `forall`

- if `F |- forall x, f(x)`, then `F |- f(t)`
  - elimination rule
  - **intuition:** if you have a way of transforming any `x` into evidence for `f(x)`, then you can use that to produce evidence for `f(t)` out of `t`

Proof with `forall`

Let's show `|- (forall x, R(x) /\ Q(x)) => (forall x, R(x)) /\ (forall x, Q(x))`

1. `forall x, R(x) /\ Q(x) |- forall x, R(x) /\ Q(x)` by assump.
2. `forall x, R(x) /\ Q(x) |- R(x) /\ Q(x)` by (1) and forall elim.
3. `forall x, R(x) /\ Q(x) |- R(x)` by (2) and /\ elim L
4. `forall x, R(x) /\ Q(x) |- forall x, R(x)` by (3) and forall intro*
5. `forall x, R(x) /\ Q(x) |- Q(x)` by (2) and /\ elim R
6. `forall x, R(x) /\ Q(x) |- forall x, Q(x)` by (5) and forall intro*
7. `forall x, R(x) /\ Q(x) |- (forall x, R(x)) /\ (forall x, Q(x))` by (4), (6) and /\ intro
8. `|- (forall x, R(x) /\ Q(x)) => (forall x, R(x)) /\ (forall x, Q(x))` by (7) and => intro.

* `x` does not occur free in LHS

Tree form

```
---------------------------------------- assump.
forall x, R(x) /\ Q(x)
  |- forall x, R(x) /\ Q(x)
---------------------------------------- forall elim
forall x, R(x) /\ Q(x)
  |- R(x) /\ Q(x)
---------------------------------------- /\ elim L
forall x, R(x) /\ Q(x)
  |- R(x)
---------------------------------------- forall intro*
forall x, R(x) /\ Q(x)
  |- (forall x, R(x))

---------------------------------------- assump.
forall x, R(x) /\ Q(x)
  |- forall x, R(x) /\ Q(x)
---------------------------------------- forall elim
forall x, R(x) /\ Q(x)
  |- R(x) /\ Q(x)
---------------------------------------- /\ elim R
forall x, R(x) /\ Q(x)
  |- Q(x)
---------------------------------------- forall intro*
forall x, R(x) /\ Q(x)
  |- (forall x, Q(x))

-------------------------------------------------------------- /\ intro
forall x, R(x) /\ Q(x)
  |- (forall x, R(x)) /\ (forall x, Q(x))
-------------------------------------------------------------- => intro
|- (forall x, R(x) /\ Q(x)) =>
   (forall x, R(x)) /\ (forall x, Q(x))
```

* **x** does not occur free in LHS

Note: bad formatting! hard to fit on slide ☹

As an OCaml program?

- OCaml's type system is not quite expressive enough to give a program whose type is that formula
  - In part, reason for that is to get good type inference
- Languages with richer type systems can do it
  - See CS 4110/6110
- Same will be true of existentials...

Evidence for `exists`

Q: What constitutes evidence for `exists x, f(x)`?

A: A *witness* object `w`, along with evidence for `f(w)`.

Proof rules for `exists`

- if `F |- f(t)` then `F |- exists x, f(x)`
  - introduction rule
  - **intuition:** if you can construct evidence for `f(t)` then `t` is a witness.

Proof rules for `exists`

- if `F |- exists x, f(x)` and `F |- f(x) => g` and **`x`** does not occur free in **`F`** or **`g`**, then `F |- g`
  - elimination rule
  - **intuition:** if you have a witness **`w`** for **`f(w)`**, and if you have a way of transforming evidence for **`f(x)`** into evidence for **`g`**, and if there are no assumptions about **`x`**, then you can use **`w`** in place of **`x`** to get evidence for **`g`**.

Proof with `exists`

Let's show `|- (exists x, Q(x) \/ R(x)) => (exists x, Q(x)) \/ (exists x, R(x))`

1. `Q(x) |- Q(x)` by assump.
2. `Q(x) |- exists x, Q(x)` by (1) and exists intro
3. `Q(x) |- (exists x, Q(x)) \/ (exists x, R(x))` by (2) and \/ intro L
4. `|- Q(x) => (exists x, Q(x)) \/ (exists x, R(x))` by (3) and => intro
5. `Q(x) \/ R(x) |- Q(x) => (exists x, Q(x)) \/ (exists x, R(x))` by (4) and weak.
6. `Q(x) \/ R(x) |- R(x) => (exists x, Q(x)) \/ (exists x, R(x))` by *repeat (1—5) with R*
7. `Q(x) \/ R(x) |- Q(x) \/ R(x)` by assump.
8. `Q(x) \/ R(x) |- (exists x, Q(x)) \/ (exists x, R(x))` by \/ elim using (7), (5), (6)
9. `|- Q(x) \/ R(x) => (exists x, Q(x)) \/ (exists x, R(x))` by (8) and => intro
10. `exists x, Q(x) \/ R(x) |- Q(x) \/ R(x) => (exists x, Q(x)) \/ (exists x, R(x))` by (9) and weak
11. `exists x, Q(x) \/ R(x) |- exists x, Q(x) \/ R(x)` by assump.
12. `exists x, Q(x) \/ R(x) |- (exists x, Q(x)) \/ (exists x, R(x))` by exists elim using (11), (10), and x does not occur free in `(exists x, Q(x) \/ R(x))` or in `(exists x, Q(x)) \/ (exists x, R(x))`
13. `|- (exists x, Q(x) \/ R(x)) => (exists x, Q(x)) \/ (exists x, R(x))` by => intro

tree form omitted; too big to fit on slides

Proof rules of IQC

Rule name	Rule
---	All rules of IPC
forall intro	if **`F |- f(x)`** and **`x`** not in FV(**`F`**) then **`F |- forall x, f(x)`**
forall elim	if **`F |- forall x, f(x)`** then **`F |- f(t)`**
exists intro	if **`F |- f(t)`** then **`F |- exists x, f(x)`**
exists elim	if **`F |- exists x, f(x)`** and **`F |- f(x) => g`** and **`x`** not in FV(**`F,g`**) then **`F |- g`**

Please hold still for 1 more minute

WRAP-UP FOR TODAY

Upcoming events

- **PS5 due Thursday**

This is logical.

THIS IS 3110