Inference Rules: Propositional Calculus

$$\frac{a \qquad b}{a \wedge b}\{\wedge I\} \qquad \frac{a \wedge b}{a}\{\wedge E_L\} \qquad \frac{a \wedge b}{b}\{\wedge E_R\}$$

$$\frac{a}{a \vee b}\{\vee I_L\} \qquad \frac{b}{a \vee b}\{\vee I_R\} \qquad \frac{a \vee b \qquad a \vdash c \qquad b \vdash c}{c}\{\vee E\}$$

$$\frac{a \vdash b}{a \to b}\{\to I\} \qquad \frac{a \qquad a \to b}{b}\{\to E\}$$

$$\frac{a}{a}\{ID\} \qquad \frac{\mathsf{False}}{a}\{CTR\} \qquad \frac{\neg a \vdash \mathsf{False}}{a}\{RAA\}$$

Some Theorems in Rule Form

$$\frac{a \wedge b}{b \wedge a}\{\wedge Comm\}$$

And Commutes

$$\frac{a \vee b}{b \vee a}\{\vee Comm\}$$

Or Commutes

$$\frac{}{a \vee (\neg a)}\{noMiddle\}$$

Law of Excluded Middle

$$\frac{a \to b \quad b \to c}{a \to c}\{\to Chain\}$$

Implication Chain Rule

$$\frac{\neg(a \vee b)}{\neg(b \vee a)}\{\neg(\vee)Comm\}$$

Not Or Commutes

$$\frac{a \to b \quad \neg b}{\neg a}\{modTol\}$$

Modus Tollens

$$\frac{a \to b}{(\neg b) \to (\neg a)}\{conPos_F\}$$

Contrapositive Fwd

$$\frac{a \quad \neg a}{False}\{+\&-\}$$

NeverBoth

$$\frac{a \to b}{(\neg a) \vee b}\{\to_F\}$$

Implication Fwd

$$\frac{(\neg a) \vee b}{a \to b}\{\to_B\}$$

Implication Bkw

More Theorems in Rule Form

$$\frac{\neg(a \lor b)}{(\neg a)\land(\neg b)}\{DeM\lor_F\}$$

DeMorgan Or Fwd

$$\frac{(\neg a)\land(\neg b)}{\neg(a \lor b)}\{DeM\lor_B\}$$

DeMorgan Or Bkw

$$\frac{\neg(a \land b)}{(\neg a)\lor(\neg b)}\{DeM\land_F\}$$

DeMorgan And Fwd

$$\frac{(\neg a)\lor(\neg b)}{\neg(a \land b)}\{DeM\land_B\}$$

DeMorgan And Bkw

$$\frac{a \lor b \qquad \neg a}{b}\{disjSyll\}$$

Disjunctive Syllogism

$$\frac{\neg(\neg a)}{a}\{\neg\ \neg_F\}$$

Double Negation Fwd

$$\frac{a}{\neg(\neg a)}\{\neg\ \neg_B\}$$

Double Negation Bkw

Some Equations of Boolean Algebra

Axioms

$a \wedge \text{False} = \text{False}$	{∧ *null*}
$a \vee \text{True} = \text{True}$	{∨ *null*}
$a \wedge \text{True} = a$	{∧ *identity*}
$a \vee \text{False} = a$	{∨ *identity*}
$a \wedge a = a$	{∧ *idempotent*}
$a \vee a = a$	{∨ *idempotent*}
$a \wedge b = b \wedge a$	{∧ *commutative*}
$a \vee b = b \vee a$	{∨ *commutative*}
$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	{∧ *associative*}
$(a \vee b) \vee c = a \vee (b \vee c)$	{∨ *associative*}
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	{∧ *distributes over* ∨}
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	{∨ *distributes over* ∧}
$\neg(a \wedge b) = (\neg a) \vee (\neg b)$	{*DeMorgan's law* ∧}
$\neg(a \vee b) = (\neg a) \wedge (\neg b)$	{*DeMorgan's law* ∨}
$\neg \text{True} = \text{False}$	{*negate True*}
$\neg \text{False} = \text{True}$	{*negate False*}
$(a \wedge (\neg a)) = \text{False}$	{∧ *complement*}
$(a \vee (\neg a)) = \text{True}$	{∨ *complement*}
$\neg(\neg a) = a$	{*double negation*}
$(a \wedge b) \rightarrow c = a \rightarrow (b \rightarrow c)$	{*Currying*}
$a \rightarrow b = (\neg a) \vee b$	{*implication*}
$a \rightarrow b = (\neg b) \rightarrow (\neg a)$	{*contrapositive*}

Theorems

$(a \wedge b) \vee b = b$	{∨ *absorption*}
$(a \vee b) \wedge b = b$	{∧ *absorption*}
$(a \vee b) \rightarrow c = (a \rightarrow c) \wedge (b \rightarrow c)$	{∨ *imp*}

Equations of Predicate Calculus

~~$(\forall x.\ f(x)) \rightarrow f(c)$ {7.3}~~
~~$f(c) \rightarrow (\exists x.\ f(x))$ {7.4}~~

$(\forall x.\ \neg f(x)) = (\neg(\exists x.\ f(x)))$ {deM ∃}
$(\exists x.\ \neg f(x)) = (\neg(\forall x.\ f(x)))$ {deM ∀}

x not free in q

$((\forall x.f(x)) \wedge q) = ((\forall x.(f(x) \wedge q))$ {∧ dist over ∀}
$((\forall x.f(x)) \vee q) = ((\forall x.(f(x) \vee q))$ {∨ dist over ∀}
$((\exists x.f(x)) \wedge q) = ((\exists x.(f(x) \wedge q))$ {∧ dist over ∃}
$((\exists x.f(x)) \vee q) = ((\exists x.(f(x) \vee q))$ {∨ dist over ∃}

$(\forall x.(f(x) \wedge g(x))) = ((\forall x.f(x)) \wedge (\forall x.g(x)))$ {∀ dist over ∧}
~~$((\forall x.f(x)) \vee (\forall x.g(x))) \rightarrow (\forall x.(f(x) \vee g(x)))$ {7.12}~~
~~$((\exists x.f(x)) \wedge (\exists x.g(x))) \rightarrow (\exists x.(f(x) \wedge g(x)))$ {7.13}~~
$(\exists x.(f(x) \vee g(x))) = ((\exists x.f(x)) \vee (\exists x.g(x)))$ {∃ dist over ∨}

$(\forall x.\ f(x)) = (\forall y.\ f(y))$ {∀R}
$(\exists x.\ f(x)) = (\exists y.\ f(y))$ {∃R}

y not free in f(x) and x not free in f(y)

Inductive Equations (axioms) and Some Theorems

sum :: Num n => [n] -> n	
sum(x: xs) = x + sum xs	sum :
sum[ ] = 0	sum[ ]
Theorem: sum = foldr (+) 0	sum.foldr
length :: [a] -> Int	
length(x: xs) = 1 + length xs	length.:
length[ ] = 0	length.[ ]
Theorem: length = foldr oneMore 0	length.foldr
(++) :: [a] -> [a] -> [a]	
(x: xs) ++ ys = x: (xs ++ ys)	++ :
[ ] ++ ys = ys	++[ ]
Theorem: xs ++ ys = foldr (:) ys xs	++.foldr
Theorem: length(xs ++ ys) = (length xs) + (length ys)	++.additive
Theorem: ((xs ++ ys) ++ zs) = (xs ++ (ys ++ zs))	++.assoc
concat :: [[a]] -> [a]	
concat(xs: xss) = xs ++ concat xss	concat.:
concat[ ] = [ ]	concat.[ ]
Theorem: concat = foldr (++) [ ]	concat.foldr
(x: [ ]) = [x]	:[ ]
(xs ≠ [ ]) = (∃x. ∃ys. (xs = (x: ys)))	(:)
(x : [x_1, x_2, …]) = [x, x_1, x_2, …]	(: …)

Patterns of Computation

Pattern: foldr (⊕) z [x_1, x_2, …, x_{n-1}, x_n] = x_1 ⊕ (x_2 ⊕ … (x_{n-1} ⊕ (x_n ⊕ z))…)

```
foldr :: (a –> b -> b) –> b –> [a] -> b
foldr (⊕) z (x: xs) = x ⊕ foldr (⊕) z xs          --foldr:
foldr (⊕) z [ ] = z                                --foldr[ ]
```

Pattern: map f [x_1, x_2, … x_n] = [f x_1, f x_2, … f x_n]

```
map :: (a –> b) –> [a] –> [b]
map f (x : xs) = (f x) : map f xs                  --map:
map f [ ] = [ ]                                    --map[ ]
```

Pattern: zipWith b [x_1, x_2, … x_n] [y_1, y_2, … y_n] =[b x_1 y_1, b x_2 y_2, … b x_n y_n]

Note: extra elements in either sequence are dropped

```
zipWith :: (a –> b –> c) –> [a] –> [b] –> [c]
zipWith b (x:xs) (y:ys) = (b x y): (zipWith b xs ys)     --zipW:
zipWith b [ ]  ys = [ ]                                  --zipW[ ]L
zipWith b xs  [ ] = [ ]                                  --zipW[ ]R
```

Pattern: iterate f x = [x, f x, f(f x), f(f(f x)), …]

```
iterate :: (a –> a) -> a –> [a]
iterate f x =  x : (iterate f (f x))               --iterate
```

Predicates, Quantifiers, and Variables

- Predicate – parameterized collection of propositions
  - P(x) is a proposition from predicate P
  - x comes from the universe of discourse, which must be specific
- ∀x.P(x) – ∀ quantifier converts predicate to proposition
  - False if and only if there is some x for which P(x) is False
- ∃x.P(x) – ∀ quantifier converts predicate to proposition
  - True if and only if there is some x for which P(x) is True
- Free and bound variables in predicate calculus formulas
  - Bound variable
    - ✓ ∀x. *e* — x is bound in the formula ∀x. *e*
    - ✓ ∃x. *e* — x is bound in the formula ∃x. *e*
  - Free variables are variables that are not bound
- Arbitrary variables in proofs
  - A free variable in a predicate calculus formula is **arbitrary** in a proof if it does not occur free in any undischarged assumption of that proof

Inference Rules: Predicate Calculus and Induction

Renaming Variables

$$\frac{F(x)\ \{x \text{ arbitrary}\}}{F(y)}\{R\}$$

$$\frac{\forall x.\ F(x) \quad \{y \text{ not in } F(x)\}}{\forall y.\ F(y)}\{\forall R\}$$

$$\frac{\exists x.\ F(x) \quad \{y \text{ not in } F(x)\}}{\exists y.\ F(y)}\{\exists R\}$$

Introducing/Eliminating Quantifiers

$$\frac{F(x)\ \{x \text{ arbitrary}\}}{\forall x.\ F(x)}\{\forall I\}$$

$$\frac{\forall x.\ F(x) \quad \{\text{universe is not empty}\}}{F(x)}\{\forall E\}$$

$$\frac{F(x)}{\exists x.\ F(x)}\{\exists I\}$$

$$\frac{\exists x.\ F(x) \quad F(x) \vdash A\ \{x \text{ not free in } A\}}{A}\{\exists E\}$$

Induction

$$\frac{P(0) \quad \forall n.(P(n) \rightarrow P(n+1))}{\forall n.P(n)}\{Ind\}$$

$$\frac{\forall n.((\forall m<n.P(m)) \rightarrow P(n))}{\forall n.P(n)}\{StrInd\}$$

Strong Induction

Principle of Mathematical Induction
another way to skin a cat

- {∀I} — an inference rule with ∀n. P(n) as it's conclusion

$$\frac{P(n)\ \{n \text{ arbitrary}\}}{\forall n.\ P(n)}\ \{\forall I\}$$

∀ Introduction

- One way to use {∀I}
  - Prove P(0)
  - Prove P(n +1) for arbitrary n
    - ✓ Takes care of P(1), P(2), P(3), …

$$\frac{P(0) \qquad \forall n. P(n) \rightarrow P(n+1)}{\forall n.\ P(n)}\ \{Ind\}$$

Induction

- Mathematical induction makes it easier
  - Proof of P(n +1) can cite P(n) as a reason
    - ✓ If you cite P(n) as a reason in proof of P(n+1), your proof relies on mathematical induction
    - ✓ If you don't, your proof relies on {∀I}
  - Strong induction makes it even easier
    - ✓ The proof of P(n+1) can cite P(n), P(n-1), … and/or P(0)

Haskell Type Specifications

- x, y, z :: Integer -- x, y, and z have type Integer
- xs, ys :: [Integer] -- sequences with Integer elements
- xy :: (Integer, Bool) -- 2-tuple with 1st component Integer, 2nd Bool
- or :: [Bool] -> Bool -- function with one argument
  argument is sequence with Bool elems
  delivers value of type Bool
- (++) :: [e] -> [e] -> [e] -- generic function with two arguments
  args are sequences with elems of same type
  type is not constrained (can be any type)
  delivers sequence with elements of
  same type as those in arguments
- sum :: Num n => [n] -> n -- generic function with one argument
  argument is a sequence with elems of type n
  n must a type of class Num
  Num is a set of types with +, *, … operations
- powerSet :: (Eq e, Show e) => Set e -> Set(Set e)
  -- generic function with one argument
  argument is a set with elements of type e
  delivers set with elements of type (Set e)
  type e must be both class Eq and class Show
  Class Eq has == operator, Show displayable