Lecture 6: Formal Syntax & Propositional Logic

CS 181O
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

First: Laziness in Haskell

Lazy Lists

```
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

complexity O(fib n) ~ $O(2^n)$

```
fibList = f 1 1
  where f a b = a : f b (a+b)
fastFib n = fibList!!n
```

complexity O(n)

```
fibs = 1:1:[ a+b | (a,b) <- zip fibs (tail fibs)]

primes = sieve [ 2.. ]
       where
         sieve (p:x) = p :
              sieve [ n | n <- x, n `mod` p > 0]
```

Monads Later

- Because Haskell is a purely functional language no function can have a side effect.
- Unfortunately input and output is a side effect period
- To cope with input and output Haskell has a new language construct known as a monad.
- We will discuss monads later in the course.

Formal Syntax and Propositional Logic

A Fragment of English

- S → NP VP
- NP → Snow White | Alice | Dorothy | Goldilocks | DET CN | DET RCN
- DET → the | every | some | no
- CN → girl | boy | princess | dwarf | giant | sword | dagger
- RCN → CN that VP | CN that NP TV
- VP → laughed | cheered | shuddered | TV NP | DV NP NP
- TV → loved | admired | helped | defeated | caught
- DV → gave

Derivation

- S ⇒ NP VP ⇒ Snow White VP
  ⇒ Snow White TV NP
  ⇒ Snow White admired NP
  ⇒ Snow White admired DET CN
  ⇒ Snow White admired the CN
  ⇒ Snow White admired the dwarf
- Draw parse tree
- Every girl admired the dwarf.

In Haskell

```
data Sent = Sent NP VP deriving Show
data NP   = SnowWhite  | Alice  | Dorothy | Goldilocks
        | NP1 DET CN | NP2 DET RCN deriving Show
data DET  = The | Every | Some | No  deriving Show
data CN   = Girl   | Boy   | Princess | Dwarf | Giant
        | Sword | Dagger deriving Show
data RCN  = RCN1 CN That VP | RCN2 CN That NP TV
        deriving Show
data That = That deriving Show
data VP   = Laughed | Cheered | Shuddered
        | VP1 TV NP | VP2 DV NP NP deriving Show
data TV   = Loved   | Admired | Helped | Defeated | Caught
        deriving Show
data DV   = Gave deriving Show
```

Example

- More details in file FSynF.hs
- "Snow White admired the dwarf" *becomes*
  - s:: Sent
  - s = Sent SnowWhite (VP1 Admired (NP1 The Dwarf))
- We will show later how to parse a sentence into a Haskell formula.

Logic

- Context free language designed for expressing Boolean-valued statements
- Translate assertions in English into logic and determine if true in model.
  - Meanings expressed in lambda calculus built from logic base.
- Start simple & work up in complexity.

Propositional Logic

- Definition of well-formed formulas of prop logic:
  - atom := p | q | r | atom'
  - F ::= atom | (¬F) | (F∨F) | (F∧F)

Use "::=" in place of "→" for productions to avoid confusion when expand

- Parens help build unique parse trees for formulas.

Propositional Logic

- Expand with abbreviations
  - $A \rightarrow B$ for $\neg A \vee B$
  - $A \leftrightarrow B$ for $(A \rightarrow B) \wedge (B \rightarrow A)$
- Often (informally) drop parentheses around terms
  - Precedence: $\neg$, $\wedge$, $\vee$, $\rightarrow$
  - $\wedge$ and $\vee$ are left associative; $\rightarrow$ is right associative.
- Sometimes add $\top$ for true and $\perp$ for false.

Induction on Formulas

- *Principle of Structured Induction*: Every formula of propositional logic has property P provided:
  - *Basic step:* every atom has property P
  - *Induction step:* if F has property P then so does $\neg F$; if F_1 and F_2 have property P then so do $(F_1 \wedge F_2)$ and $(F_1 \vee F_2)$.

Balanced Parens

- Every propositional formula F has equal numbers of left and right parentheses. Moreover every proper prefix of F has more left parentheses then right parentheses.
- *Proof*:
  - *Basic step:* An atom has 0 left parentheses and 0 right parentheses, so they are equal. It also has no proper prefixes.

Balanced Parens (2)

- *Proof (cont.)*:
  - *Induction step:* S'pose F has = parens, then so does $(\neg F)$. If F_1 and F_2 each have = parens then so do $(F_1 \wedge F_2)$ and $(F_1 \vee F_2)$.
  - Show for prefixes on board in class
- *Proof in text for Prop 4.3 is incomplete. Can you see why?*

Semantics of Propositional Logic

- Meaning of formula depends on meaning of propositional letters.
  - Start with valuation fcn V: Prop Letters → {true,false}
  - Extend to V^+: Prop Logic Formulas → {true,false} by
    - $V^+(p) = V(p)$ if p is propositional letter
    - $V^+(\neg\phi)$ = false iff $V^+(\phi)$ = true
    - $V^+(\phi\vee\gamma)$ = true iff $V^+(\phi)$ = true or $V^+(\gamma)$ = true (or both)
    - $V^+(\phi\wedge\gamma)$ = true iff $V^+(\phi)$ = true and $V^+(\gamma)$ = true
    - $V^+(\phi\rightarrow\gamma)$ = false iff $V^+(\phi)$ = true and $V^+(\gamma)$ = false

Truth Tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	*T*	F	T	T	T	T
T	*F*	F	F	T	F	F
F	*T*	T	F	T	T	F
F	*F*	T	F	F	T	T

Each row corresponds to different valuation

Categories of WFFs

- A formula ϕ is *valid*, or a *tautology*, if for all valuations V, we have $V^{+}(\phi)$ = true.
- A formula ϕ is *satisfiable* if for some valuation V, we have $V^{+}(\phi)$ = true.
- A formula ϕ is *contingent* if for some valuation V, we have $V^{+}(\phi)$ = false.
- A formula ϕ is *unsatisfiable*, or a *contradiction*, if for all valuations V , we have $V^{+}(\phi)$ = false.

Semantic Entailment

- $\phi_1, \ldots, \phi_n \vDash \psi$ iff for every valuation V s.t. $V^*(\phi_1) = \ldots = V^*(\phi_n)$ = true, then $V^*(\psi)$ = true
  - Example: $P \vDash Q \rightarrow P$
  - Read $\phi_1, \ldots, \phi_n \vDash \psi$ as $\phi_1, \ldots, \phi_n$ *logically implies* ψ
- Hence, $\vDash \psi$ iff ψ is a tautology.
- Show: $\phi_1, \ldots, \phi_n, \phi \vDash \psi$ iff $\phi_1, \ldots, \phi_n \vDash \phi \rightarrow \psi$

Questions?