Computational Semantics with Haskell

Yulia Zinova

Winter 2016/2017

We follow Van Eijck and Unger 2010, electronic access from the library

Overview

- We will talk about some example languages:
  - languages for playing simple games
  - logical languages
  - fragments of programming languages
  - fragments of natural language
- When we will be dealing with the semantics of natural languages, we will use predicate logic.
- As a preparation, we will have a look at the propositional and predicate logic: how they can be used to represent the meaning of natural language sentences and how to implement their syntax in Haskell.
- Download this file:
  `http://www.computational-semantics.eu/FSynF.hs`

Sea Battle

- Rules:
  1. 2 players
  2. 2 grids per player, each with 10 x 10 fields: 1 – 10 and A – J
  3. players do not see each others' grids
  4. at the beginning, each player distributes their ships over one of the grids
  5. fleet: a battleship (5 squares), a frigate (4 squares), two submarines (3 squares), a destroyer (2 squares).
  6. the grid with ships is also used to record enemy shots
  7. the other grid is used to record shots fired at the enemy

Sea Battle: Grammar

- **column** → *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J*
- **row** → *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10*
- **attack** → **column row**
- **ship** → *battleship* | *fregate* | *submarine* | *destroyer*
- **reaction** → *missed* | *hit* **ship** | *sunk* **ship** | *lost_ battle*
- **turn** → **arrack reaction**

Exercise: revise the grammar in such a way that it is explicit that the game ends once one of the players is defeated.

Mastermind

- *Mastermind* is a code-breaking game for two players
- Code-maker decides on a row of coloured pegs (fixed set of colours)
- Code-breaker tries to guess the color pattern
- Each turn: codebreaker names a sequence; codemaker replies with *black* for each correct colour-place combination and with *white* for each correct colour in the wrong place.
- Goal :find out the sequence

Mastermind: Grammar

- **colour** $\rightarrow$ *red* | *yellow* | *green* | *lila* | *blue* | *orange*
- **answer** $\rightarrow$ *black* | *white*
- **guess** $\rightarrow$ **colour colour colour colour**
- **reaction** $\rightarrow$ **{answer}**
- **turn** $\rightarrow$ **guess reaction**
- **game** $\rightarrow$ **turn | turn game**

Exercise: revise the grammar in order to guarantee that a game has at most 4 turns
Exercise: change the definition of **reaction** to ensure that the grammar generates a finite language

Grammars for Games: Exercises

- Write the grammar for chess.
- Write the grammar for Bingo!
- Bingo rules:
  - A bingo ticket is a card with a 5x5 grid. 5 columns on the card correspond to 5 letters of the name of the game "B-I-N-G-O".
  - 24 numbers per each card are random from the limits of 1 to 75. The center of the card is left empty.
  - B column: from 1 to 15, I column: from 16 to 30, N column: from 31 to 45, G column: from 46 to 60, O column: from 61 to 75
  - Round: the caller selects a random number and calls it. All the players mark it on their tickets.
  - The winner is determined when one or several of the players complete the winning bingo pattern.

A fregment of English

- We want to write rules for English sentences like the following
- *The girl laughed.*
- *No dwarf admired some princess that shuddered.*
- *Every girl some boy loved cheered.*
- *The wizard that helped Snow White defeated the giant.*
- We need rules for: subject-predicate structure of sentences, internal structure of noun phrases, common nouns with and without relative clauses.
- Let us write the grammar!

A language of talking about classes

- Consider the following interaction engine for an inference engine (program the handles interaction with a knowledge base):
- Questions (or queries) are of the form: Are all PN PN? Are no PN PN? Are any PN PN? Are any PN not PN? What about PN?
- Statements are of the form: All PN are PN. No PN are PN. Some PN are PN. Some PN are not PN.
- PN = plural noun
- We will later provide a semantics for this fragment so that it could be used.

Propositional logic

- No we will look at a grammar for *propositional logic*, where we use p, q, r, p', q', r', p'', q'', r'', ... to indicate atomic propositions
- $atom \rightarrow p \mid q \mid r \mid$ **atom**
- **F** $\rightarrow$ **atom** $\mid \neg F \mid$ **(F** $\vee$ **F)** $\mid$ **(F** $\wedge$ **F)**

Principle of structural induction

- If you need to prove that every formula of propositional logic has property P, you need to use induction
- **Induction base:** Every atom has property P
- **Induction step:** If F has property P, so does $\neg F$, if F_1 and F_2 have property P, then so do $(F_1 \vee F_2)$ and $(F_1 \wedge F_2)$
- Exercise: Show that every propositional formula has an equal number of left and right parenthesis
- Exercise: Show that propositional formulas have only one parse tree

Making life easier

- The ‘official’ way of writing propositional formulas is a bit clumsy
- We will use p_2 instead of p''
- We will often omit parenthesis when it does not result in ambiguity (conjunction and disjunction)
- 2 abbreviations: *implication* and *equivalence*:
- Implication: write $F_1 \rightarrow F_2$ for $\neg(F_1 \wedge \neg F_2)$
- Equivalence: write $F_1 \leftrightarrow F_2$ for $(F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$

Translating from natural language to propositional logic

- *If it rains and the sun is shining, then there will be a rainbow.*
- *The wizard polishes his hand and learns a new spell, or he is lazy.*
- *The wizard will deal with the devil only if he has a plan to outwit him.*
- *If neither unicorns nor dragons exist, then neither do goblins.*
- *You can either have ice cream or candy floss, but not both.*
- Define a connective $\oplus$ for exclusive disjunction using the already defined connectives.

Polish notation

- Formulas of propositional logic can be written without parenthesis, if we use prefix or postfix notation.
- Prefix notation is also called Polish notation.
- $\mathbf{F} \rightarrow \textbf{atom} \mid \neg\ F \mid \vee FF \mid \wedge FF$
- Exercise: translate $\wedge \vee pqr$ into infix notation
- Exercise: use the principle of structural induction to prove that formulas of propositional logic in infix notation are uniquely readable

Haskell implementation

- Exercise: Implement a function countOperations for computing a number of operations in the formula
- Exercise: Implement a function listAtoms that collects the names of propositional atoms that occur in a formula.

Predicate logic

- In propositional logic, the following two sentences will be not related:
  1. *Every prince saw a lady*
  2. *Some prince saw a beautiful lady*
- To capture the internal structure of such sentences, we need predicate logic.

Predicate logic aka first-order (predicate) logic

- Predicate logic is an extension of propositional logic with *structured basic propositions* and *quantifications*:
  1. A structured basic proposition consists of an *n*-ary predicate followed with *n* variables.
  2. A universally quantified formula consists of the symbol $\forall$ followed by a variable followed by a formula.
  3. An existentially quantified formula consists of the symbol $\exists$ followed by a variable followed by a formula.
  4. Other ingredients are as in propositional logic

Predicate logic: definition

- Definition in assumption that predicates have arity not more than 3:
  v $\rightarrow$ *x* | *y* | *z* | **v'**
  P $\rightarrow$ *P* | **P'**
  R $\rightarrow$ *R* | **R'**
  S $\rightarrow$ *S* | **S'**
  atom $\rightarrow$ **P v** | **R v v** | **S v v v**
  F $\rightarrow$ **atom** | **(v = v)** | $\neg$ **F** | **F** $\wedge$ **F** | **F** $\vee$ **F** | $\forall$ **v F** | $\exists$ **v F**
- Poll! `http://directpoll.com/r?XDbzPBd3ixYqg8pA3St08d1irQ6lHS0WJlPc1h1i`

Bound variables

- In a formula $\forall x F$ (or $\exists x F$), the quantifier occurrence binds all occurrences of x in F that are not bound by an occurrence of $\forall x$ or $\exists x$ inside F.
- Syntactic definition:an occurrence of $\forall x$ or $\exists x$ in a formula F binds an occurrence of x in F if in the syntax tree for F *the occurrence* $\forall x$ *(or* $\exists x$) c-commands x, and inside F there are no other occurrences of $\forall x$ or $\exists x$ that c-command x.
- A predicate logic formula is called *open* if it contains at least one variable occurrence which is free. If all variable occurrences are bound, the formula is called *closed*/a predicate logical *sentence*.

Predicate logic

- Exercise: write a formula that represents the following sentences:
  1. *Some prince saw a beautiful lady.*
  2. *Every prince saw a lady.*

Predicate logic formulas in Haskell

- We will combine predicates with lists of variables $\rightarrow$ flexible arity

  v $\rightarrow x \mid y \mid z \mid$ **v'**

  vlist $\rightarrow$ [] $\mid$ **v: vlist**

  P $\rightarrow P \mid$ **P'**

  atom $\rightarrow$ **P vlist**

  F $\rightarrow$ **atom** $\mid$ **v** $=$ **v** $\mid \neg$ **F** $\mid \wedge$ **Flist** $\mid \vee$ **Flist** $\mid \forall$ **v F** $\mid \exists$ **v F**

  Flist $\rightarrow$ [] $\mid$ **F: Flist**

Predicate logic formulas in Haskell: Exercises

- Write a function `sentence` that checks whether a formula is a sentence.
- Write a function `noNegImpl` that replaces each formula by an equivalent one without occurrences of `Impl` and `Neg`

-

-

References:

Van Eijck, J. and Unger, C. (2010). *Computational semantics with functional programming*. Cambridge University Press.