COMS22201: 2015/16

Language Engineering (Semantics)

Dr Oliver Ray
(csxor@Bristol.ac.uk)

Department of Computer Science
University of Bristol

Tuesday 9th February, 2016

Feedback from Class Test

- 2 questions each worth 6 marks
- Scores ranged between 2 and 11 (out of 12)
- Most common errors included
  - leaving the fact $x=x_0$ out of the invariant
  - leaving the fact $y \geq 1$ out of the invariant
  - not explaining the notation x_0 (this is possibly a bit harsh, but since all sorts of conventions are in use why not be polite and define your notation)
  - leaving out type constructors from Haskell definitions
- Both questions were part of the Lab Sheet – discussed next

Week 15 Lab: Question 1

1. Consider the following grammar which defines a bit as being either on or off and a word as being any non-empty sequence of bits:

$$\langle bit \rangle \quad ::= \quad \text{'on'} \mid \text{'off'}$$
$$\langle word \rangle \quad ::= \quad \langle bit \rangle \mid \langle bit \rangle \ \langle word \rangle$$

   (a) Define a Haskell type `Bit` which has two possible values `On` and `Off`. Ensure your type automatically derives membership of the `Eq`, `Ord`, `Enum`, `Show` and `Read` classes.

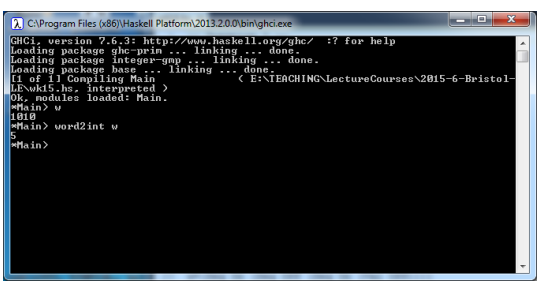
   (b) Define a recursive Haskell type `Word` which represents a sequence of `Bits`. Instantiate your type into the `Show` class by writing a function `show` that turns the word representing 'on off on off' into the string "1010", for example.

   (c) Assuming words represent unsigned binary numbers with the *least* significant bit on the left, write a Haskell function that computes the integer value represented a given word (which would be 5 for the word illustrated above).

Week 15 Lab: Question 1

```
data Bit = On | Off
  deriving (Eq,Ord,Enum,Show,Read)

data Word = Val Bit | Seq Bit Word

instance Show Word where
  show (Val On) = "1"
  show (Val Off) = "0"
  show (Seq On w) = "1"++(show w)
  show (Seq Off w) = "0"++(show w)

word2int (Val On) = 1
word2int (Val Off) = 0
word2int (Seq On w) = 1+2*(word2int w)
word2int (Seq Off w) = 0+2*(word2int w)

w::Word
w=(Seq On (Seq Off (Seq On (Val Off))))
```

Week 15 Lab: Question 1



Week 15 Lab: Question 2

2. Consider the following grammar defining the arithmetic expressions, Boolean expressions and program statements of the language **While** in terms of numerals and variables:

$$\begin{aligned}
\langle Aexp \rangle \quad &::= \quad \langle Num \rangle \mid \langle Var \rangle \\
&\mid \quad \langle Aexp \rangle \text{ '+' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ '*' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ '-' } \langle Aexp \rangle \\
\langle Bexp \rangle \quad &::= \quad \text{'true'} \mid \text{'false'} \\
&\mid \quad \langle Aexp \rangle \text{ '=' } \langle Aexp \rangle \mid \langle Aexp \rangle \text{ '}\leq\text{' } \langle Aexp \rangle \\
&\mid \quad \text{'}\neg\text{' } \langle Bexp \rangle \mid \langle Bexp \rangle \text{ '}\wedge\text{' } \langle Bexp \rangle \\
\langle Stm \rangle \quad &::= \quad \langle Var \rangle \text{ ':=' } \langle Aexp \rangle \mid \text{'skip'} \mid \langle Stm \rangle \text{ ';' } \langle Stm \rangle \\
&\mid \quad \text{'if' } \langle Bexp \rangle \text{ 'then' } \langle Stm \rangle \text{ 'else' } \langle Stm \rangle \\
&\mid \quad \text{'while' } \langle Bexp \rangle \text{ 'do' } \langle Stm \rangle
\end{aligned}$$

Week 15 Lab: Question 2

(a) Define two Haskell types `Num` and `Var` as synonyms for Haskell's native `Integer` and `String` types, respectively.

(b) Define three Haskell types `Aexp`, `Bexp` and `Stm` to represent the syntactic categories of **While**.

Note that you may have to use the qualified syntax `Main.Num` for numerals to avoid a conflict with Haskell's native `Num` class.

(c) Represent the following program as a Haskell term:

```
z:=x; while (2≤y) do (z:=z*x; y:=y-1)
```

Week 15 Lab: Question 2

```
type Num = Integer

type Var = String

data Aexp   = N Main.Num | V Var
  | Add Aexp Aexp | Mult Aexp Aexp | Sub Aexp Aexp
   deriving (Show, Eq, Read)

data Bexp = TRUE | FALSE
  | Eq Aexp Aexp | Le Aexp Aexp
  | Neg Bexp | And Bexp Bexp
   deriving (Show, Eq, Read)

data Stm = Ass Var Aexp | Skip | Comp Stm Stm
  | If Bexp Stm Stm
  | While Bexp Stm
   deriving (Show, Eq, Read)
```

Week 15 Lab: Question 2

```
z:=x; while (2≤y) do (z:=z*x; y:=y-1)
```

```
p::Stm

p =
  (Comp
   (Ass "z" (V "x"))
    (While
     (Le (N 2) (V "y"))
     (Comp
      (Ass "z" (Mult (V "x") (V "x")))
      (Ass "y" (Sub (V "y") (N 1)))
     )
    )
  )
```

Week 15 Lab: Question 3

```
z:=x; while (2≤y) do (z:=z*x; y:=y-1)
```

3. Use a loop invariant to prove the program above computes the value of x raised to the power of (the initial value of) y for all initial $x, y > 0$.

Week 15 Lab: Question 3

Add Pre- and Post-Conditions:

```
// pre: x=x0 > 0 and  y=y0 > 0
z = x ;
while (2 ≤ y) do (
      z = z * x ;
      y = y - 1
)
// post: z = x0 ^ y0
```

Trace loop for some particular values:

	0	1	2	3
x	2	2	2	2
Y	4	3	2	1
z	2	4	8	16

Determine a loop invariant:

$z.x \hat{} (y-1) = x_0 \hat{} y_0$ and $y \geq 1$

Week 15 Lab: Question 3

Check that the invariant

$z.x \hat{} (y-1) = x_0 \hat{} y_0$ and $y \geq 1$

Satisfies (I will add solutions below before the next lecture)

(a) Initialisation:

…

(b) Termination:

…

(c) Maintenance

…

Week 15 Lab: Question 3

Add Pre- and Post-Conditions:

```
// pre: x=x0 > 0 and  y=y0 > 0
z = x ;
while ¬(1 = y) do (
      z = z * x ;
      y = y - 1
)
// post: z = x0 ^ y0
```

Trace loop for some particular values:

	0	1	2	3
x	2	2	2	2
Y	4	3	2	1
z	2	4	8	16

Determine a loop invariant:

$z = x \wedge (y_0 - y + 1)$ and ~~$y \neq 1$~~ and $x = x_0$

Week 14 Lab: Question 3

3. Prove for all $n > 0$ there is a legal English sentence of the form

   Buffalo^n.

   In other words prove there is an infinite sequence of grammatically correct sentences of the form

   Buffalo. Buffalo buffalo. Buffalo buffalo buffalo. ...

   Hint: the word 'buffalo' can be a noun (i.e. a bison-like animal), an adjectival noun (i.e. relating to the city of Buffalo in the state of New York), or a verb (meaning to bully or intimidate)!

Week 14 Lab: Question 3

- The first point to note is that the required proof can only be conducted with respect to a formal grammar describing (at least some subset of) the English language. Our first step is to define a suitable grammar; and for this we can use the one from last week's lecture slides.
- Then we can prove the theorem by induction on the length n of the sentence. Our next step is to work out the trick that will make this possible. It is not difficult to see that we can use relative clauses to increase the length of a Buffalo sentence by 2 – which suggests we can work separately on even and odd length sentences.
- Although the base cases will turn out to be very easy, we do need to be very precise about the induction hypothesis and to formalise exactly what it is we are trying to prove. So our final step will be to pin this down and complete the proof.

Week 14 Lab: Question 3

It suffices to start with the following (fragment of) English grammar :

S	→	NP | NP VP	Sentence	
NP	→	N | NP RC | NA N	Noun Phrase	
VP	→	V NP	Verb Phrase	
RC	→	NP V	Relative Clause	
NA	→	PN	Noun Adjunct	
V	→	buffalo	Verb	(to bully)
N	→	buffalo	Noun	(a bison)
PN	→	Buffalo	Proper Noun	(NY state)

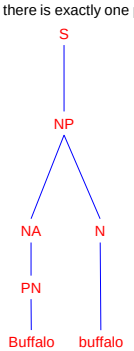
And then prove the theorem by induction on the length n of the sentence:

Week 14 Lab: Question 3

For n=1 there is exactly one parse:

Bison

For n=2 there is exactly one parse:



NewYork bison

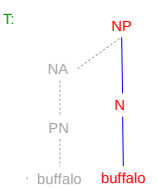
Week 14 Lab: Question 3

We will now formally show that for all n>0 there is a legal parse tree of the sentence buffalo^n in which at least one buffalo is a Noun.

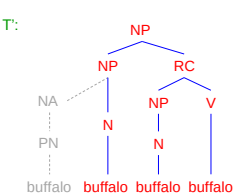
BASE CASES (n=1 and n=2): Both cases already shown by the parse trees above.

INDUCTIVE STEP (n=k for some k>2): Assume there is a legal parse tree of the sentence buffalo^{k-2} in which at least one buffalo is a Noun. Then, because N only occurs in NP there must be a sub-tree of the form T below (with or without an NA). But we can replace any such subtree by the corresponding tree T' below to give a legal parse tree of the sentence buffalo^k in which at least one buffalo is a Noun.

T:



T':

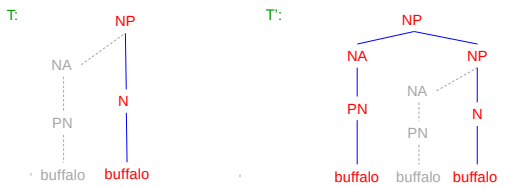


Week 14 Lab: Question 3

In theory it is possible to construct a proof with a single base case using Noun Adjuncts to increase the length of a sentence by 1.

BASE CASES (n=1): Already shown by the parse tree above.

INDUCTIVE STEP (n=k for some k>1): Assume there is a legal parse tree of the sentence buffalo^{k-1} in which at least one buffalo is a Noun. Then, because N only occurs in NP there must be a sub-tree of the form T below (with or without an NA). But we can replace any such subtree by the corresponding tree T' below to give a legal parse tree of the sentence buffalo^{k} in which at least one buffalo is a Noun.



Week 14 Lab: Question 3

- But it could be argued the latter proof is less convincing because it relies on sentences that are not intuitively convincing from semantic point of view. While "NewYork bison" are obviously bison from the state of NewYork it is less clear what "NewYork NewYork bison" are (perhaps NewYork bison not in another state, but this gets tenuous).
- On the other hand the former proof uses very logical sentences (especially if we always replace the rightmost noun in the sentence) that mean things like "bison that are bullied by bison that other bison bully".

Week 14 Lab: Question 4

4. Observe for all $n > 0$ there is a legal English sentence of the form

   A white male (whom a white male)^n (hired)^n hired a white male.

   Use this fact to prove that English is *not* a regular language.

Week 14 Lab: Question 4

- It is important to note is this result can't be shown by simply providing a non-regular subset of English.

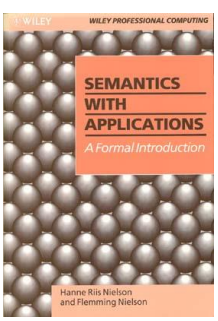
  That would demonstrate nothing because it is well known that every non-finite language has infinitely many non-regular sub-languages by simple counting arguments!
- Your approach should be show that set of L sentences of this form is a non-regular language N which is the intersection of English with a regular language R.

  So you need to find the regular language R and show the non-regularity of N. Then the result follows by contradiction using the closure of regular languages under intersection.
- For the non-regularity proof you could consider using the Pumping Lemma.

  But this won't work here if it is believed that the adjective can be indefinitely repeated (e.g. "A white white male" could mean a "A very very white male")?

  If so, then you would need to reason directly about regular automata.

Course Textbook: Nielson



Semantics with Applications: A Formal Introduction

H. Nielson and F. Nielson

Revised online edition (click link below for pdf)

https://www.cs.bris.ac.uk/Teaching/Resources/COMS22201/nielson.pdf

Course Topics: Chapters 1, 4, 6 and 2