innovate achieve lead

BITS Pilani
Pilani Campus

CS/IS F214
Logic in Computer Science

MODULE: PROPOSITIONAL LOGIC

Proofs by Induction: Structural Induction:

- Motivating Example
- Proof Principle
- Relation to Mathematical Induction
- Example and Exercises

Structural Induction – Motivating Example

- Definition:
  - A ***binary tree*** is a tree in which *every node has at most two children.*
- Exercise:
  - Using induction, prove that:
    - **For any binary tree T, *N(T) = L(T) + 1***
      - where **N(T)** is the number of nodes in **T** and
      - **L(T)** is the number of links / edges in **T**.
- Note that:
  - A binary tree is structured as
    - a root node and
    - one or two sub-trees

Structural Induction – Example Problem 1: Proof

- Induction Basis:
  - Consider a binary tree **T** with only the root.
    - ***N(T) = 1*** and ***L(T) = 0***
- Inductive Cases: There are two cases:
  1. a binary tree **T** with a root, and one sub-tree **T1** or
  2. a binary tree **T** with a root, and two sub-trees **T1** and **T2**

Structural Induction – Example Problem 1: Proof [2]

- Inductive Case 1:

  Consider a binary tree **T** with a root, and one sub-tree **T1**

- Induction Hypothesis:
  - Assume that the property is true for (any given) sub-tree **T1**

- Induction Step:
  - ***N(T) = N(T1) + 1*** ***(+1 is for root node)***

    = L(T1) + 2 ***(by hypothesis)***
  - ***L(T) = L(T1) + 1***

    i.e.
    - ***N(T) = L(T) + 1***

Structural Induction – Example Problem 1: Proof [3]

- Inductive Case 2:

  Consider a binary tree **T** with a root, and two sub-trees **T1** and **T2**

- Induction Hypothesis:
  - Assume that the property is true for (any two given) sub-trees **T1** and **T2**

- Induction Step:
  - $N(T) = N(T1) + N(T2) + 1 = L(T1) + L(T2) + 3$ ***(by hypotheses)***
  - $L(T) = L(T1) + L(T2) + 2$
  - i.e.
    - $N(T) = L(T) + 1$

Structural Induction – Proof Principle

- Proof Principle:
  - Let **P** be a property applicable on a structure **S** that is constructed out of:
    - base (atomic) cases $\mathbf{B_0, B_1, \ldots B_k}$
    - inductive cases:
      - $\mathbf{R_1(S_{1,0}, S_{1,1}, \ldots S_{1,r1})}$
      - $\mathbf{R_2(S_{2,0}, S_{2,1}, \ldots S_{2,r2})}$
      - **...**
      - $\mathbf{R_m(S_{m,0}, S_{m,1}, \ldots S_{m,rm})}$
  - If
    - **P** is true for each of the base cases $\mathbf{B_0, B_1, \ldots B_k}$
    - *for each inductive case* $\mathbf{R_j}$:

    P is true for $\mathbf{S_{j,0}, S_{j,1}, \ldots S_{j,rj}}$ implies **P** is true for $\mathbf{R_j(S_{j,0}, S_{j,1}, \ldots S_{j,rj})}$
  - then
    - **P** is true for all **S**

Structural Induction - Exercises

- Exercise:
  1. Look back at the proof on binary trees:
     - *identify the property* **P**
     - *identify the structure **S** with its bases cases and recursive cases*.
  2. Argue that Mathematical Induction is a special case of Structural Induction:
     - *express natural numbers as a structure*

Structural Induction – Proof Principle

- Proof Principle:
  - Let **P** be a property applicable on a structure **S** that is constructed out of:
    - base (atomic) cases $\mathbf{B_0, B_1, \dots B_k}$
    - inductive cases:
      - $\mathbf{R_1(S_{1,0}, S_{1,1}, \dots S_{1,r1})}$
      - $\mathbf{R_2(S_{2,0}, S_{2,1}, \dots S_{2,r2})}$
      - **...**
      - $\mathbf{R_m(S_{m,0}, S_{m,1}, \dots S_{m,rm})}$
  - If
    - **P** is true for each of the base cases $\mathbf{B_0, B_1, \dots B_k}$
    - *for each inductive case* $\mathbf{R_j}$:

    P being true for $\mathbf{S_{j,0}, S_{j,1}, \dots S_{j,rj}}$ implies **P** is true for $\mathbf{R_j(S_{j,0},S_{j,1},\dots S_{j,rj})}$
  - then
    - **P** is true for all **S**

Structural Induction – Example 2 : Matching Parentheses

- A string is said to have matching parentheses if
  - *every left parenthesis is matched by a corresponding right parenthesis occurring after it* (somewhere to the right) and
  - *every right parenthesis is matched by a corresponding left parenthesis occurring before it* (somewhere to the left)

- e.g.
  - **(((a) (b)))** *is a string with matching parentheses*
  - **(a))(** *is not a string with matching parentheses*

Matching Parenthesis: Context

- Verifying whether a string has matching parentheses is often a requirement
  - e.g. when *a program is compiled* :
    - a compiler verifies whether the *program is syntactically well-formed* before it is translated

- For the sake of simplifying the discussion
  - we will ignore all other characters – *other than the left and the right parentheses* – and refer to *strings of matching parentheses*

Matching Parentheses - Grammar

- Given the following grammar (**Gr-MP**)

  1. **S ---> e**
  2. **S ---> (S)**
  3. **S ---> SS**

prove that all strings generated by **Gr-MP** *have matching parentheses*:

  - **Note**: **e** denotes the empty string i.e. *string of length* **0**.

- Exercise:
  - Note that the definition is inductive:
    - *identify the base case and the inductive cases .*

Matching Parentheses: Proof by Structural Induction

Grammar (**Gr-MP**):

1. **S ---> e**
2. **S ---> (S)**
3. **S ---> SS**

Prove that *all strings generated by* **Gr-MP** *have matching parentheses*.

Matching Parentheses: Proof by Structural Induction

Grammar (**Gr-MP**):

1. **S ---> e**
2. **S ---> (S)**
3. **S ---> SS**

Prove that *all strings generated by* **Gr-MP** *have matching parentheses.*

Proof:

- **Induction Basis:**
  - S ---> e
    - The empty string has matching parentheses (trivially)
- **Induction Step:**
  - Case 2: S ---> (S)
  - Case 3: S ---> SS

Matching Parentheses: Proof by Structural Induction *(contd.)*

Grammar (**Gr-MP**):

1. **S ---> e**
2. **S ---> (S)**
3. **S ---> SS**

Prove that all strings generated by **Gr-MP** have matching parentheses.

Case 2: S_1 ---> (S_2)

- **Induction Hypothesis:**
  - Assume S_2 has matching parentheses.
- **Induction Step:**
  - Then S_1 generates (S_2) where
  - *the left-most left-parenthesis matches with the right-most right-parenthesis* and
  - *in the rest of the string i.e. in* S_2
    - all parentheses have already been matched (**by hypothesis**).

Matching Parentheses: Proof by Structural Induction *(contd.)*

Grammar (**Gr-MP**):

1. **S ---> e**
2. **S ---> (S)**
3. **S ---> SS**

Prove that all strings generated by **Gr-MP** have matching parentheses.

Case 3: S_1 ---> S_2 S_3

- **Induction Hypothesis:**
  - Assume S_2 and S_3 have matching parentheses.
- **Induction Step:**
  - Then S_1 generates S_2 S_3 where
    - *all parentheses in substring* S_2 *have already been matched* (**by assumption**) and
    - *all parentheses in substring* S_3 *have already been matched* (**by assumption**)

Matching Parentheses – Proof of correctness

- Exercise:

  Prove that *any string of matching parentheses can be generated by the grammar* **Gr-MP.**

  [Hint: *Prove by induction on the length of a string*. **End of Hint.]**

Matching Parentheses – Generalization

Consider a generalized version of strings of matching parentheses:

where not just parentheses,

i. but *braces* (i.e. '{', and '}'), and *brackets* (i.e. '[', and ']') as well must be matched and

ii. they may be nested.

e.g.

- **((([])))**{()[]}** is well-formed but
- **{(})** is not well-formed

Matching Parentheses – Generalization

Exercise:

1. Define a grammar (say **Gr-MPg**) to generate/validate strings of matching parentheses (*according to the generalized version in the previous slide*).
2. Repeat the proofs of correctness i.e. argue that
   i. *any string generated by* **Gr-MPg** *has matching parentheses* and
   ii. **Gr-MPg** *generates all such strings*

Matching Parentheses – Generalization and Text

- Consider the addition of other characters (*alphabetic*, *numeric*, and other *punctuation / operator symbol*) to strings of matching parentheses (generalized to include braces and brackets):
  - i.e. **(((a[i]+b)*c)/d){((*f)())[100]);}** is a well-formed string of matching parentheses.
- Exercises:
  - Modify grammar **Gr-MPg** to grammar **Gr-MPgt**
    - that generates (or recognizes) all well-formed strings of matching parentheses (generalized) with other characters allowed.
  - Argue that the proofs on **Gr-MPg** (see Ex. 2(i) and 2(ii) in the previous slide) are applicable for **Gr-MPgt**.

Matching Parentheses – Program Syntax

- Provide valid examples of nesting of parentheses, braces, and brackets inside each other in a C program:
  - nesting of () inside {} and nesting of [] inside {}
  - nesting of [] inside () and nesting of () inside []
  - nesting of () inside () and nesting of [] inside []
  - nesting of {} inside {}
- Question:
  - Can {} ever be nested inside () or inside [] ?

Matching Parentheses – Program Syntax - Exercises

- Exercise:
  - Modify grammar **Gr-MPgt** to incorporate permissible nesting in a program in C (or in your *favorite language*).

- Study a grammar specifying the syntax of a program in C (or in your *favorite language*):
  - In what ways positioning of text with respect to (generalized) parentheses is restricted?
    - For instance, is a sequence of the form $\alpha(\beta)\lambda[\varphi]$ possible in a C program?
      - α, β, λ, and φ are strings not containing a left/right parenthesis, a left/right bracket, or a left/right brace.
    - What about a sequence of the form $\alpha(\beta)\lambda\{\varphi\}$?

Exercise – Grammar for DNF

- Prove that
  - any formula generated by the grammar **Gr-PropL-OE-3** (see Lecture 12) can be
    - rewritten into an equivalent formula in canonical form (say DNF)
    - that can be generated by the grammar **Gr-PropL-DNF** (see Lecture 13b).
- [**Hint**: Use structural induction on **Gr-PropL-OE-3:**
  - Prove that --> can be eliminated
  - Prove that the increasing order of precedence ($\vee$, $\wedge$, and $\neg$) can be maintained i.e. *parentheses are not needed*.

End of Hint.]