BITS Pilani
Pilani Campus

CS/IS F214
Logic in Computer Science

MODULE: PROPOSITIONAL LOGIC

Syntax – Order of Evaluation

RECALL: Syntax and Order of Evaluation

- One way to eliminate ambiguity in the grammar of a language is
  - to insist that the user (i.e. one who writes sentences) *must eliminate all ambiguity by parenthesizing everything*
- Alternatively, one can capture precedence and associativity rules in the grammar!

Propositional Logic: Typical Order of Evaluation

- The following *precedence* and *associativity* are conventional in **propositional logic**:
  - --> has the lowest precedence
  - $\vee$ has the next higher precedence
  - $\wedge$ has the next higher precedence
  - $\neg$ has the highest precedence

  - --> is right-associative
    - What about $\vee$ and $\wedge$?

Grammar – Approach 2

- Capturing precedence and associativity rules in the grammar:

  - --> has the lowest precedence
  - ∨ has the next higher precedence
  - ∧ has the next higher precedence
  - ¬ has the highest precedence
  - all operators are right-associative

Grammar – Approach 2

Rules of the grammar

(**Gr-PropL-OE-2**): *incomplete*

1. **Form ---> DisForm '-->' Form**
2. **Form ---> DisForm**

We want to capture precedence and associativity rules:

--> has the lowest precedence and is right-associative

These two rules state that

any formula (Form) is in one of two forms

1. a *<u>disjunctive formula</u>* (**DisForm**), followed by the symbol '-->', and by another formula (**Form**)
2. only a *<u>disjunctive formula</u>* (**DisForm**)

where a disjunctive formula *<u>does not include an implication</u>*.

(**Gr-PropL-OE-2**): *incomplete*

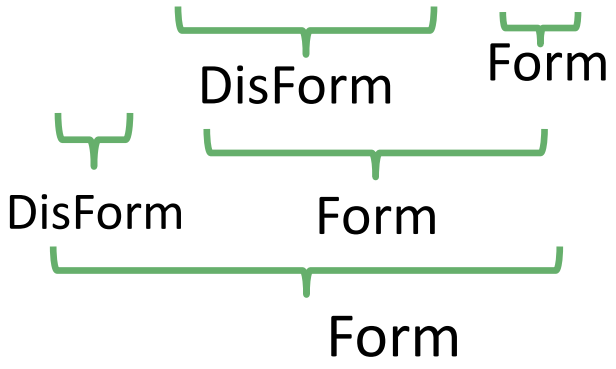
1. **Form ---> DisForm '-->' Form**
2. **Form ---> DisForm**

i.e. if *a formula includes* '-->' then *we separate the formula into*

(i) a left sub-formula that does not contain an '-->' **and**

(ii) a right sub-formula

e.g. **p --> q ∨ r ∧ s --> t**



Recursive Rule – Sentences Generated

- Give a rule of the form:
  - X --> a X

  what are the sentences that can be generated?
- Given a pair of rules of the form:
  - X --> a X
  - X --> a

  what are the sentences that can be generated?

Rules of the grammar

Precedence rules:

--> has the lowest precedence

∨ has the next higher precedence

both are right-associative

(Gr-PropL-OE-2): *incomplete*

1. **Form ---> DisForm '-->' Form**
2. **Form ---> DisForm**
3. **DisForm ---> ConForm '∨' DisForm**
4. **DisForm ---> ConForm**

The last two rules state that

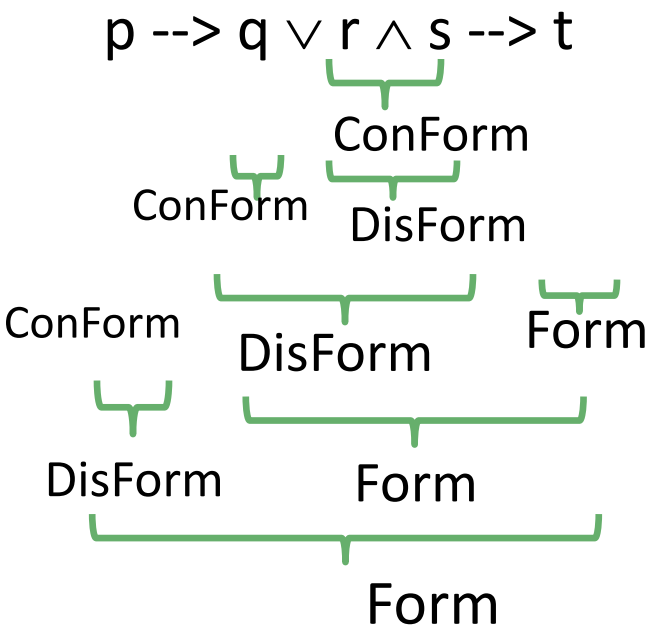
any disjunctive formula (DisForm) is in one of two forms

3. *a conjunctive formula* (**ConForm**), followed by disjunction symbol ('∨'), and then by *a disjunctive formula* (**DisForm**)

4. *only a conjunctive formula* (**ConForm**)

where a conjunctive formula does not include a disjunction (i.e. '∨')

Rules of the grammar



p --> q ∨ r ∧ s --> t

ConForm

ConForm

DisForm

ConForm

DisForm

Form

DisForm

Form

Form

(Gr-PropL-OE-2): *incomplete*

1. **Form ---> DisForm '-->' Form**
2. **Form ---> DisForm**
3. **DisForm ---> ConForm '∨' DisForm**
4. **DisForm ---> ConForm**

Rules of the grammar

(**Gr-PropL-OE-2**): *incomplete*

1. Form ---> Disform '-->' Form
2. Form ---> DisForm
3. DisForm ---> ConForm '∨' DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm '∧' ConForm
6. ConForm ---> NegForm

Grammar – Approach 2

We capture precedence and associativity rules:

--> has the lowest precedence

V has the next higher precedence

∧ has the next higher precedence

All three operators are right-associative

Rules of the grammar

(**Gr-PropL-OE-2**):

1. Form ---> Disform ‘-->’ Form
2. Form ---> DisForm
3. DisForm ---> ConForm ‘∨’ DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm ‘∧’ ConForm
6. ConForm ---> NegForm
7. NegForm --> ‘¬’ NegForm
8. NegForm --> p

where p is any propositional atom

Grammar – Approach 2

We capture precedence and associativity rules:

--> has the lowest precedence

∨ has the next higher precedence

∧ has the next higher precedence

¬ has the highest precedence

All the binary operators are right-associative

Rules of the grammar

(**Gr-PropL-OE-2**):

1. Form ---> Disform '-->' Form
2. Form ---> DisForm
3. DisForm ---> ConForm '∨' DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm '∧' ConForm
6. ConForm ---> NegForm
7. NegForm ---> '¬' NegForm
8. NegForm ---> p

where p *is any propositional atom*

Grammar – Approach 2

We capture precedence and associativity rules:

--> has the lowest precedence

∨ has the next higher precedence

∧ has the next higher precedence

¬ has the highest precedence

All the binary operators are right-associative

Question: What is the drawback of the approach/grammar?

Question : Are there formulas that cannot be generated using this grammar? Is this question relevant?

Approach 2 - Limitations

While this grammar captures a set of precedences (¬ over ∧, ∧ over ∨, ∨ over -->), there are limitations:

1. Precedence cannot be overruled

   i.e. ¬p ∧ q is interpreted as

   (NOT p) AND q

   but there is *no way to generate the form* NOT (p AND q)

Grammar – Approach 2

(**Gr-PropL-OE-2**):

1. Form ---> Disform '-->' Form
2. Form ---> DisForm
3. DisForm ---> ConForm '∨' DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm '∧' ConForm
6. ConForm ---> NegForm
7. NegForm ---> '¬' NegForm
8. NegForm ---> p

where p is any propositional atom

Approach 2 - Limitations

Issue: *Precedence cannot be overruled*

i.e. **¬p ∧ q** is interpreted as

(NOT p) AND q

Solution: NOT (p AND q)

can be generated from this grammar using rule 8

Grammar – Approach 2A

(**Gr-PropL-OE-3**):

1. Form ---> Disform '-->' Form
2. Form ---> DisForm
3. DisForm ---> ConForm '∨' DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm '∧' ConForm
6. ConForm ---> NegForm
7. NegForm ---> '¬' NegForm
8. NegForm ---> '(' Form ')
9. NegForm ---> p

where p *is any propositional atom*

Exercise

- Argue that *rule* 8 allows: *precedence to be over-ruled by parenthesizing sub-expressions*.
- For instance, generate / parse formulas
  - $\neg(p \wedge q)$ and
  - $((p \dashrightarrow q) \vee r) \wedge s)$

  using this grammar.

Grammar – Approach 2A

(Gr-PropL-OE-3):

1. Form ---> Disform ‘-->’ Form
2. Form ---> DisForm
3. DisForm ---> ConForm ‘∨’ DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm ‘∧’ ConForm
6. ConForm ---> NegForm
7. NegForm ---> ‘¬’ NegForm
8. NegForm ---> ‘(‘ Form ‘)
9. NegForm ---> p

where p *is any propositional atom*

Propositional Logic – Grammar – Approach 2A.

(**Gr-PropL-OE-3**):

1. Form ---> Disform '-->' Form
2. Form ---> DisForm
3. DisForm ---> ConForm '$\vee$' DisForm
4. DisForm ---> ConForm
5. ConForm ---> NegForm '$\wedge$' ConForm
6. ConForm ---> NegForm
7. NegForm ---> '$\neg$' NegForm
8. NegForm ---> '(' Form ')
9. NegForm ---> p

   where p *is any propositional atom*

Exercise: *Write formulas that cannot be generated from this grammar .* OR *Prove that this grammar can generate all well-formed-formulas.*

[Hints:

- Use **Gr-PropL-AMB** as the "correct" definition of *well-formed formulas.*
- Use induction to prove:
  - *all formulas generated by **Gr-PropL-AMB** can be generated by **Gr-PropL-OE-3** with appropriate parentheses.* **End of Hints.]**