Propositional Logic - Syntax and Parsing

HE Mingxin, Max

I2ML(H) Spring 2023 (CS104|CS108)

Topic 3.1

Propositional logic - Syntax

Syntax

We need a quick method of identifying if a group of symbols is a logical argument.

We usually define a syntax.

Example 3.1

Grammar of Chinese; Grammar of English; Syntax of Java.

Let us define syntax for propositional logic.

Propositions

The logic is over a list of propositions.

- Sky is blue.
- Sun is hot.
- … *many more*

We do not care what each one says. We give each one of them a symbol.

Propositional variables

We assume that there is a countably-infinite set Vars of propositional variables.

- Since Vars is countable, we assume that variables are indexed.

$$\text{Vars} = \{\ p_1,\ \ p_2,\ \ \ldots\ \ \}$$

- The variables are just names/symbols without inherent meaning

- We may also use $p, q, r, .., x, y, z$ to denote the propositional variables

- Propositional variables are also called Boolean variables

Commentary: All results presented in this course are extendable to uncountable Vars. For the uncountable setting, we need transfinite induction. We will ignore those extensions.

Logic connects the variables

A logical argument connects the propositions.

Let us list all the possible ways of connecting them.

True and false

We should be able to talk about

- always true statement
- always false statement

Example 3.2

- *An apple is an apple.* *always true*
- *I like Apple and I do not like Apple.* *always false*

Logical connectives: Not, And, and Or

We may also need ability to say

- a statement that says negation of another
- two statements are true at the same time
- at least one of the two statements are true

Example 3.3

- *The apple is not sweet.*
- *The apple is sweet and* NY *is far.*
- *The apple is sweet or* NY *is far.*

More logical connectives: Implies, equality, and disequality

We may also need ability to say

- Implication
  if a statement is true then some other statement is also true
- Equivalence
  truth value of two statements are same
- Disequality
  truth value of two statements are different
  - Usually called exclusive or, meaning exactly one of the two is true

Example 3.4

- *If I work then I make money.* *(implication)*
- *I like an apple if and only if I like a pen.* *(equivalence)*
- *A is here or B is here, but both are not here.* *(exclusive or)*

Logical connectives

The following 10 symbols are called logical connectives. (may have less or more)

formal name	symbol	read as	
true	$\top$	top	0-ary symbols
false	$\bot$	bottom	
negation	$\neg$	not	unary symbols
conjunction	$\wedge$	and	
disjunction	$\vee$	or	
implication	$\Rightarrow$	implies	binary symbols
equivalence	$\Leftrightarrow$	bi-implication; iff	
exclusive or	$\oplus$	xor	
open parenthesis	(		punctuation
close parenthesis	)		

We assume that the logical connectives are not in Vars.

Propositional formulas

A propositional formula is a finite string containing symbols in Vars and logical connectives.

Definition 3.1

The set of propositional formulas is the smallest set P *such that*

- $\top, \bot \in \mathrm{P}$
- *if* $p \in \mathrm{Vars}$ *then* $p \in \mathrm{P}$
- *if* $F \in \mathrm{P}$ *then* $\neg F \in \mathrm{P}$
- *if* $\circ$ *is a binary symbol and* $F, G \in \mathrm{P}$ *then* $(F \circ G) \in \mathrm{P}$

Some notation

Definition 3.2
$\top, \bot$, *and* $p \in \mathsf{Vars}$ *are* *atomic formulas.*

Definition 3.3
For each $F \in \mathsf{P}$, *let* $\mathsf{Vars}(F)$ *be the set of variables appearing in* F.

Another Definition (TextB)

Propositional Logic Symbols

Three types of symbols in propositional logic:

- Propositional variables: p, q, r, p_1, etc.
- Connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.
- Punctuation: (and).

Definition of well-formed formulas

Let $\mathcal{P}$ be a set of propositional variables. We define the set of well-formed formulas over $\mathcal{P}$ inductively as follows.

1. A propositional variable in $\mathcal{P}$ is well-formed.
2. If α is well-formed, then $(\neg\alpha)$ is well-formed.
3. If α and β are well-formed, then each of $(\alpha \wedge \beta), (\alpha \vee \beta), (\alpha \rightarrow \beta), (\alpha \leftrightarrow \beta)$ is well-formed.

One more Definition (TextE)

$\mathcal{L}_0$-Formulas

Definition 4

The **propositional language** $\mathcal{L}_0$ is the smallest set L such that L is a set of finite sequences of symbols in

$$S_0 = \{(,), \neg, \to\} \cup \{A_n \mid n \in \mathbb{N}\}.$$

and such that

1. $\langle A_n \rangle \in L$, for each $n \in \mathbb{N}$. [1]
2. If $s \in L$, then $(\neg s) \in L$.
3. If $s, t \in L$, then $(s \to t) \in L$.

---

[1] $\langle A \rangle$ denote the length-1 sequence that consists of only one symbol A.

Other connectives

Other connectives $\vee$, $\wedge$, $\leftrightarrow$ are treated as abbreviations of formulas (involving $\{\neg, \rightarrow\}$ only) as follows:

$$\begin{array}{rcl} p \vee q & \text{iff} & \neg p \rightarrow q \\ p \wedge q & \text{iff} & \neg(p \rightarrow \neg q) \\ p \leftrightarrow q & \text{iff} & (p \rightarrow q) \wedge (q \rightarrow p) \end{array}$$

The following sets of connectives are complete:

- $\{\neg, \wedge, \vee\}$ and $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, by NDF
- $\{\neg, \rightarrow\}$,
- $\{\neg, \wedge\}$ and $\{\neg, \vee\}$.

Prove that $\{\vee\}$, $\{\wedge\}$, $\{\rightarrow\}$, $\{\neg, \leftrightarrow\}$ are not complete.

Another list of binary connectives:

Symbol	Equivalent	Remarks
$\wedge$	$p \wedge q$	且
$\vee$	$p \vee q$	或
$\rightarrow$	$p \rightarrow q$	如果 p 就 q
$\leftarrow$	$p \leftarrow q$	如果 q 就 p
$\leftrightarrow$	$p \leftrightarrow q$	p 当且仅当 q
$\oplus$	$(p \vee q) \wedge \neg(p \wedge q)$	异或，相当于对称差
$\uparrow$	$\neg(p \wedge q)$	与非
$\downarrow$	$\neg(p \vee q)$	或非
$<$	$\neg p \wedge q$	相当于 $q \setminus p$
$>$	$p \wedge \neg q$	相当于 $p \setminus q$

Show that $\{\uparrow\}$ and $\{\downarrow\}$ are complete.

Examples of propositional formulas

Exercise 3.1

Which of the following are in P*?*

- $\top \Rightarrow \bot$ ✗
- $(\top \Rightarrow \bot)$ ✓
- $(p_1 \Rightarrow \neg p_2)$ ✓
- (p_1)✗
- $\neg\neg\neg\neg\neg\neg\neg\neg\neg p_1$ ✓

Not all strings over Vars and logical connectives are in P.

We need a method to recognize a string belongs to P or not.

How can we argue that a string does or does not belong to P?

Commentary: Please carefully look at the generation grammar. We need to carefully understand the role of parenthesis to disambiguate formulas. It is an interesting note that $\neg$ does not need parenthesis.

Topic 3.2

Encoding arguments into logic

Example : symbolic argument

Example 3.5

We have seen the following argument.

If c then if s then f. not f. Therefore, if s then not c.

where

- c = *the seed catalogue is correct*
- s = *seeds are planted in April*
- f = *the flowers bloom in July*

We can write the above argument as propositional formula as follows

$$\Big(\big(\underbrace{(c \Rightarrow (s \Rightarrow f))}_{\textit{Premise 1}} \wedge \underbrace{\neg f}_{\textit{Premise 2}} \big) \Rightarrow \underbrace{(s \Rightarrow \neg c)}_{\textit{Conclusion}} \Big)$$

Topic 3.3

Parsing formulas

Parse tree

$F \in \mathrm{P}$ iff F is obtained by unfolding of the generation rules

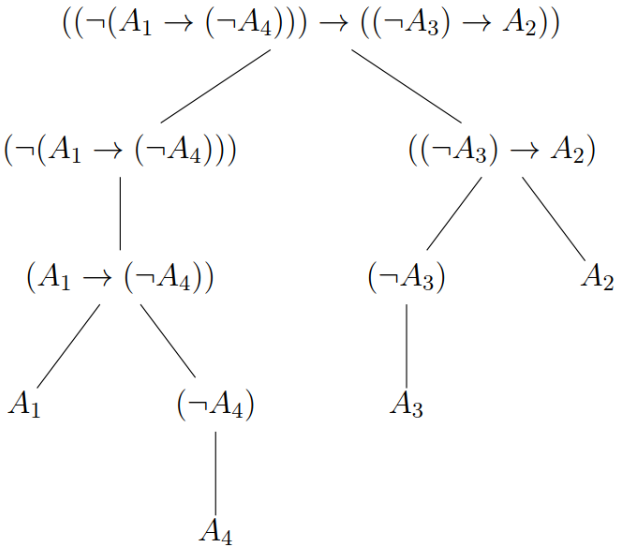
Definition

A parse tree of a formula $F \in \mathrm{P}$ is a tree such that

- *the root is F,*
- *leaves are atomic formulas, and*
- *each internal node is formed by applying some formation rule on its children.*

Example

$$
\begin{tikzcd}
 & (p_1 \Rightarrow (\neg p_2 \Leftrightarrow (p_1 \wedge p_3))) \arrow[dl] \arrow[dr] & & & \\
p_1 & & (\neg p_2 \Leftrightarrow (p_1 \wedge p_3)) \arrow[dl] \arrow[dr] & & \\
 & \neg p_2 \arrow[d] & & (p_1 \wedge p_3) \arrow[dl] \arrow[dr] & \\
 & p_2 & p_1 & & p_3
\end{tikzcd}
$$


$((\neg(A_1 \to (\neg A_4))) \to ((\neg A_3) \to A_2))$
$(\neg(A_1 \to (\neg A_4)))$
$((\neg A_3) \to A_2)$
$(A_1 \to (\neg A_4))$
$(\neg A_3)$
A_2
A_1
$(\neg A_4)$
A_3
A_4

Parse tree and unique parsing

Theorem 3.1

$F \in \mathrm{P}$ iff there is a parse tree of F.

Proof.

The reverse direction is immediate. In the forward direction, we prove a stronger theorem, i.e., existence of unique parsing tree. □

Theorem 3.2

Each $F \in \mathrm{P}$ has a unique parsing tree.

Proof.

The proof is at the last section of the slides. □

Commentary: The proof is needed for the rigor of logic. However, we skipped the proof in the class.

Parse tree is a directed-acyclic graph (DAG)

We have been thinking that the parsing produces parse tree.

However, the parsing produces a parse DAG.

Example

Consider formula $(p_1 \vee (\neg p_1 \wedge p_2))$. The following is the parse tree of the above formula.

$$\begin{tikzcd}
 & (p_1 \vee (\neg p_1 \wedge p_2)) \arrow[dr] \arrow[dddl] & & \\
 & & (\neg p_1 \wedge p_2) \arrow[dl] \arrow[dr] & \\
 & \neg p_1 \arrow[dl] & & p_2 \\
\color{red}{p_1} & & &
\end{tikzcd}$$

Subformula

Definition 3.5

A formula G is a subformula of formula F if G occurs within F. G is a proper subformula of F if $G \neq F$. Let $sub(F)$ denote the set of subformulas of F.

The nodes of the parse tree of F form the set of subformulas of F.

Definition 3.6

Immediate subformulas are the children of a formula in its parse tree, and leading connective is the connective that is used to join the children.

Example 3.9

$$sub((\neg p_2 \Leftrightarrow (p_1 \land p_3))) = \{(\neg p_2 \Leftrightarrow (p_1 \land p_3)), \neg p_2, (p_1 \land p_3), p_1, p_2, p_3\}$$

The leading connective of F is $\Leftrightarrow$.

Commentary: *Note that the above definition does not allow $p_2 \Leftrightarrow (p_1 \land p_3)$ to be a subformula of F, because $p_2 \Leftrightarrow (p_1 \land p_3)$ is not a formula. In later discussions, we may drop parenthesis in our writings and it may cause confusion. So, when we apply the above definition we need to keep the invisible parentheses in our mind.*

Topic 3.4

Shorthands

Too many parentheses

In the above syntax, we need to write a large number of parentheses.

Using precedence order over logical connectives, we may drop some parentheses without losing the unique parsing property.

Example 3.10

Consider $((p \land q) \Rightarrow (r \lor p))$

- *We may drop outermost parenthesis without any confusion*

$$(p \land q) \Rightarrow (r \lor p)$$

- *If* $\land$ *and* $\lor$ *get precedence over* $\Rightarrow$ *during parsing, we do not need the rest of parentheses*

$$p \land q \Rightarrow r \lor p$$

Precedence order

We will use the following precedence order in writing the propositional formulas

Precedence order

We will use the following precedence order in writing the propositional formulas

$$\begin{tikzcd}
 & & \neg \arrow[dll, dash] \arrow[d, dash] \arrow[drr, dash] & & \\
\lor \arrow[dr, dash] \arrow[drrr, dash] & & \land \arrow[dl, dash] \arrow[dr, dash] & & \oplus \arrow[dlll, dash] \arrow[dl, dash] \\
 & \Leftrightarrow & & \Rightarrow &
\end{tikzcd}$$

Using precedence order

Consider the following formula for $n > 1$

$$F_0 \circ_1 F_1 \circ_2 F_2 \circ_3 \cdots \circ_n F_n,$$

where F_0,..,F_n are either atomic or enclosed by parentheses, or their negation.

We transform the formula as follows

- Find an $\circ_i$ such that $\circ_{i-1}$ and $\circ_{i+1}$ have lower precedence if they exist.
- Introduce parentheses around $F_{i-1} \circ_i F_i$ and call it $F'_i \triangleq (F_{i-1} \circ_i F_i)$.

$$F_0 \circ_1 \cdots \circ_{..} F_{i-2} \circ_{i-1} F'_i \circ_{i+1} F_{i+1} \circ_{..} \cdots \circ_n F_n$$

We apply the above until $n = 1$ and then apply the normal parsing.

Inside of F_is may also have ambiguities, which are recursively resolved using the above procedure.

Commentary: We have not presented the above as a formal algorithm. However, we can present the procedure in the above style. You will find many computer science texts do not write their algorithms in a formal presentation to avoid cumbersome notation. Please learn to handle.

Example: parsing using the precedence order

Example 3.11

Consider formula $p \wedge q \Rightarrow r \vee p$*. Let us try to bring back the parentheses.*

$\Rightarrow$ *has lower precedence than* $\wedge$*, therefore we can group neighbours of* $\wedge$

$$(p \wedge q) \Rightarrow r \vee p$$

Since $\vee$ *has higher precedence over* $\Rightarrow$*, we first group* $\vee$*.*

$$(p \wedge q) \Rightarrow (r \vee p)$$

Now we can group $\Rightarrow$ *without any confusion*

$$((p \wedge q) \Rightarrow (r \vee p))$$

Example precedence order

Example 2.12

Which of the following formulas can be unambiguously parsed?

- $\neg p \lor (p \oplus q) \Leftrightarrow p \land q$ ✓
- $p \lor q \land r$ ✗
- $p \lor q \lor r$ ✗
- $p \Rightarrow q \Rightarrow r$ ✗

Associativity preference may further reduce the need of parenthesis

Associative

Problem: If a binary operator repeats, we do not know how to group.

Solution: we give preference to one side or another.

Let us make all our operators "right associative", i.e., first group the rightmost occurrence.

Example 3.13

Consider formula $p \Rightarrow q \Rightarrow r$.

We first group the right $\Rightarrow$: $p \Rightarrow (q \Rightarrow r)$

Then, we group the left $\Rightarrow$: $(p \Rightarrow (q \Rightarrow r))$

Commentary: Not all operators are affected by associativity. For example, $\land$ and $\lor$ operators have same meaning if we use any order of associativity. On the other hand, $\Rightarrow$ needs a convention for associativity.

Exercise 3.2

Modify the parsing procedure of the earlier slide to support the above.

Substitution

Definition 3.7

For $F \in \mathrm{P}$ and $p_1, \ldots, p_k \in \mathrm{Vars}$, let $F[G_1/p_1, \ldots, G_k/p_k]$ denote another formula obtained by simultaneously replacing all occurrence of p_i by a formula G_i for each $i \in 1..k$.

Example 3.14

1. $(p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p] = ((r \oplus s) \Rightarrow (r \Rightarrow (r \oplus s)))$
2. $(p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p, x/r] \neq (p \Rightarrow (r \Rightarrow p))[(r \oplus s)/p][x/r]$ **!!!**

Exercise 3.3

a. Definition 3.7 *is informal. Give a formal definition.*

b. Write the result of substitutions in the second example.

c. Give a most general restriction on substitutions such that simultaneous and sequential substitutions produce the same result.

Notation for substitution

For shorthand, we may write a formula F as

$$F(p_1, \dots, p_k),$$

where we say that variables $p_1, \dots, p_k$ play a special role in F.

$$\text{Let } F(G_1, \dots, G_n) \text{ be } F[G_1/p_1, \dots, G_k/p_k].$$

Example 3.15

Let $F(p, q) = \neg p \oplus q$

$F(r \lor q, \top) = \neg(r \lor q) \oplus \top$

Commentary: This notation is very useful in the case when we do not know F but want to talk about the substitutions in a convenient way.

Topic 3.5

Problems

Exercise: symbolizing bad and good puzzle**

Exercise 3.4

People are either good or bad. The good people always tell the truth and the bad people always tell a lie. Now let us consider the following puzzle.

There are two people A and B. A said some thing, but we could not hear. B said, "A is saying that A is bad". What are A and B?

Encode the above puzzle into a propositional logic formula.

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan. It is difficult to encode because we also need to model internal mental state of the participants.

Exercise: more puzzles

Exercise 3.5

People are either good or bad. The good people always tell the truth and the bad people always tell a lie. Now let us consider the following puzzle.

There are three people A, B, and C. A said, "All of us are bad.". B said, "Exactly one of us is good.". What are A, B, and C?

Encode the above puzzle into a propositional logic formula.

Commentary: The puzzle is borrowed from What is The Name of This Book? by Raymond M. Smullyan

Let expression

We may extend the grammar of proportional logic with let expressions.

$$(\text{let } p = F \text{ in } G)$$

Let-expression is a syntactic device to represent large formulas succinctly.

$$(\text{let } p = F \text{ in } G) \text{ represents } G[F/p]$$

Example 3.16

$$(\mathit{let}\ p = (q \wedge r)\ \mathit{in}\ ((p \wedge s) \vee (q \Rightarrow \neg p))) \qquad \mathit{represents} \qquad ((q \wedge r) \wedge s) \vee (q \Rightarrow \neg(q \wedge r))$$

Exercise 3.6

Give an example in which let expressions can represent a formula in exponentially less space.

Precedence order

Exercise 3.7

Add minimum parentheses in the following formulas such that it has unique parsing under our precedence order

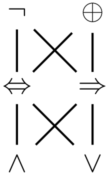
1. $p \land q \lor r \land s \land t \lor u \lor v \land w$
2. $p \Rightarrow \neg q \oplus p \lor p \land \neg r \Leftrightarrow s \land t$

Commentary: Please work the above problems with and without associative preference rules. In the exams, we will make it clear.

Custom precedence order

Exercise 3.8

Consider the following precedence order



Add minimal parentheses in the following formulas such that they have unique parsing tree

1. $\neg p \Rightarrow q \wedge r \Rightarrow p \Rightarrow q$
2. $p \Rightarrow \neg q \oplus p \vee p \wedge \neg r \Leftrightarrow s \wedge t$

Commentary: *Please work the above problems with and without associative preference rules. In the exams, we will make it clear.*

Topic 3.6

Extra lecture slides: unique parsing

Properties of well-formed formulas

Theorem Template:

Theorem: For every well-formed propositional formula φ, $P(\varphi)$ is true.

Induction over natural numbers

Let the natural numbers start from 0. Let P be some property. We want to prove that every natural number has property P.

Theorem: $P(0)$, $P(1)$, $P(2)$, ..., are all true.

Proof.

Base case: Prove $P(0)$.

Induction step: Consider an arbitrary $k \geq 0$. Assume that $P(k)$ is true. Prove that $P(k+1)$ is true.

By the principle of mathematical induction, $P(n)$ is true for $n = 1, 2, 3, \ldots$.

Another Definition (TextB)

Propositional Logic Symbols

Three types of symbols in propositional logic:

- Propositional variables: p, q, r, p_1, etc.
- Connectives: $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$.
- Punctuation: (and).

Definition of well-formed formulas

Let $\mathcal{P}$ be a set of propositional variables. We define the set of well-formed formulas over $\mathcal{P}$ inductively as follows.

1. A propositional variable in $\mathcal{P}$ is well-formed.
2. If α is well-formed, then $(\neg\alpha)$ is well-formed.
3. If α and β are well-formed, then each of $(\alpha \land \beta), (\alpha \lor \beta), (\alpha \rightarrow \beta), (\alpha \leftrightarrow \beta)$ is well-formed.

A structural induction template for well-formed formulas

Theorem: For every well-formed formula φ, $P(\varphi)$ holds.

Proof by structural induction:

Base case: φ is a propositional variable q. Prove that $P(q)$ holds.

Induction step:

Case 1: φ is $(\neg a)$, where a is well-formed.
Induction hypothesis: Assume that $P(a)$ holds.
We need to prove that $P((\neg a))$ holds. (¬a is not well formed.)

Case 2: φ is $(a * b)$ where a and b are well-formed and $*$ is a binary connective.
Induction hypothesis: Assume that $P(a)$ and $P(b)$ hold.
We need to prove that $P((a * b))$ holds.

By the principle of structural induction, $P(\varphi)$ holds for every well-formed formula φ. QED

For the formulas defined in Text B.

Matching parentheses

Theorem 3.3

Every $F \in \mathrm{P}$ has matching parentheses, i.e., equal number of '(' and ')'.

Proof.

base case:

atomic formulas have no parenthesis. Therefore, matching parenthesis

induction steps:

We assume $F, G \in \mathrm{P}$ has matching parentheses.

Let n_F and n_G be the number of '(' in F and G respectively.

Trivially, $\neg F$ has matching parentheses. (¬F is well formed.)

For some binary symbol $\circ$, the number of both '(' and ')' in $(F \circ G)$ is $n_F + n_G + 1$.

Due to the structural induction, the property holds. $\square$

Prefix of a formula

Theorem 3.4

A proper prefix of a formula is not a formula.

Proof.

We show a proper prefix of a formula is in one of the following forms.

1. strictly more '(' than ')',
2. a (possibly empty) sequence of $\neg$.

Clearly, both the cases are not in P.

base case:

A proper prefix of atomic formulas is empty string, which is the second case

Exercise 3.9

Give examples of the above two cases

Prefix of a formula II

Proof(contd.)

induction step:
Let $F, G \in \mathrm{P}$.

Consider proper prefix F' of $\neg F$. There are two cases.

- $F' = \epsilon$, case 2
- $F' = \neg F''$, where F'' is a proper prefix of F. Now we again have two subcases for F''.
  - If F'' is in case 1, F' belongs to case 1
  - If $F'' = \neg..\neg$, F' belongs to case 2

...

Prefix of a formula III

Proof(contd.)

By induction F and G have balanced parenthesis.

Consider proper prefix H of $(F \circ G)$, F' be prefix of F, and G' be prefix of G.

- If $H = (F \circ G$, H belongs to case 1 because H has one extra '('
- If $H = (F \circ G'$, H belongs to case 1 (why?)

Similarly the following cases are handled

- $H = (F\circ$
- $H = (F$
- $H = (F'$
- $H = ($

$\square$

Exercise 3.10

Complete the (why?).

Unique parsing

Theorem 3.5

Each $F \in \mathrm{P}$ has a unique parsing tree.

Proof.

$\nu(F) \triangleq$ number of logical connectives in F. We apply induction over $\nu(F)$.

base case: $\nu(F) = 0$

F is an atomic formula, therefore has a single node parsing tree.

inductive steps: $\nu(F) = n$

We assume that each F' with $\nu(F') < n$ has a unique parsing tree.

case $F = \neg G$: Since G has a unique parsing tree, F has a unique parsing tree.

case $F = (G \circ H)$:

Suppose there is another formation rule such that $F = (G' \circ' H')$.

Since $F = (G \circ H) = (G' \circ' H')$, $G \circ H) = G' \circ' H')$.

Wlog, G is prefix of G'.

Since $G, G' \in \mathrm{P}$, G can not be proper prefix of G'. Therefore, $G = G'$.

Therefore, $\circ = \circ'$. Therefore, $H = H'$. Therefore, only one way to unfold F.

F has a unique parsing tree. $\square$

Parsing algorithm

Algorithm 3.1: PARSER

Input: F : a string over Vars and logical connectives
Output: parse tree if $F \in \mathrm{P}$, exception FAIL otherwise

if $F = p$ *or* $F = \top$ *or* $F = \bot$ **then** **return** $(\{F\}, \emptyset)$;
if $F = \neg G$ **then**
  $(V, E) := \textsc{Parser}(G)$;
  return $(V \cup \{F\}, E \cup \{(F, G)\})$;
if *F has matching parentheses and $F = (F')$* **then**
  $G :=$ smallest prefix of F' where non-zero parentheses match or atom after a sequence of '$\neg$'s;
  $o'H := tail(F', len(G))$;
  if *the above two match succeed* **then**
    $(V_1, E_1) := \textsc{Parser}(G)$;
    $(V_2, E_2) := \textsc{Parser}(H)$;
    return $(V_1 \cup V_2 \cup \{F\}, E_1 \cup E_2 \cup \{(F, G), (F, H)\})$;
Throw FAIL

Commentary: The previous proofs suggest a parsing algorithm to generate parsing tree.

End of Lecture 3