Discrete Mathematics (II) Spring 2012

Lecture 5: Formation Tree and Parsing Algorithm

Lecturer: Yi Li

1 Overview

In this lecture, we completely focus on syntax of proposition. And the term "proposition" does always mean well-defined proposition without explicit declaration.

The most important topic is to make sure the unique readability of a well defined proposition. And then a proposition is associated with a tree, called formation tree. Therefore, we find the properties of a well-defined proposition based on its tree form. Finally, we introduce a parsing algorithm to check the validity of a sequence of symbols, with a formation tree as a by-product.

2 Some properties of proposition

2.1 Ambiguity

In everyday language, we sometimes could speak some sentence which could be understood totally with different meaning, which is called ambiguous.

Example 1. *Consider the following sentences:*

1. *The lady hit the man with an umbrella.*
2. *He gave her cat food.*
3. *They are looking for teachers of French, German and Japanese.*

Once you read these sentences, you way wonder:

1. Is the lady using an umbrella to hit or is she hitting a man who is carrying an umbrella?
2. Is he giving cat food to her or is he giving her cat some food?
3. Are they looking for teachers who can each teach one language or all three languages?

You may find more examples. Here, the ambiguity is not caused by the meaning/semantics of some word. The reason is just the way to group words in different approaches.

Similarly, you could encounter the same trouble when you write a piece of segment of code. The example is the way to nest if and else. You can also consider the following expression "$4/2 \times 2$".

Without the help of parentheses, proposition could also run into the same problem. Consider the following example.

Example 2. *Consider the following proposition*

$$A_1 \vee A_2 \wedge A_3,$$

which is definitely not well-defined.

Solution. We have two possible different propositions

1. $(A_1 \vee A_2) \wedge A_3$
2. $A_1 \vee (A_2 \wedge A_3)$

Of course, they have different abbrevation truth tables. □

Mathematics is rigid. One of the responsibilities of mathematical logic is to eliminate ambiguity from mathematics. So we should first eliminate ambiguity in mathematical logic.

2.2 Parentheses

Theorem 1. *Every well-defined proposition has the same number of left as right parentheses.*

Proof. Consider the symbols without parentheses first.

And then prove it by induction with more complicated propositions according to the construction of a complicated proposition. □

Here, we actually use the depth of nested connectives, which will be introduced in the next section.

Theorem 2. *Any proper initial segement of a well defined proposition contains an excess of left parenthesiss. Thus no proper initial segement of a well defined propositon can itself be a well defined propositions.*

Go back to lecture three if your forget what is a proper initial segment.

Proof. Prove it by induction from simple to complicated propositions. And we also need Theorem 1. □

3 Formation Tree

A proposition is a sequence of symbols. The structure determines the reading of a proposition. In this section, we are to map a proposition to a tree, named formation tree. It will help us to read a proposition.

Definition 3 (Top-down)**.** *A formation tree is a finite tree T of binary sequences whose nodes are all labele with propositions. The labeling satisfies the following conditions:*

1. *The leaves are labeled with propositional letters.*
2. *if a node σ is labeled with a proposition of the form $(\alpha \vee \beta), (\alpha \wedge \beta), (\alpha \to \beta)$ or $(\alpha \leftrightarrow \beta)$, its immediate successors, $\sigma\hat{}0$ and $\sigma\hat{}1$, are labeled with α and β (in that order).*
3. *if a node σ is labeled with a proposition of the form $(\neg\alpha)$, its unique immediate successor, $\sigma\hat{}0$, is labeled with α.*

How to map a well-defined proposition to a tree? Consider the following example.

Example 3. *The formation tree of $(A \vee B), ((A \wedge B) \to C)$.*

It is easy to draw its formation tree for this example. However, when a proposition is complicated enough, it is not easy. Just rethink your approach. We may follow the procedure of constructing a proposition from a propositional letters in a bottom-up way, even we define it in a top-down approach.

These two examples are specified. What about the other propositions? Given proposition, we have the following questions:

1. Is there a formation tree?
2. Is it unique?

It is lucky that we have the following Theorem.

Theorem 4. *Each well-defined proposition has a unique formation tree associated with it.*

Proof. (Sketch)

1. Existence of the formation tree by induction on depth.
2. Uniqueness of the formation tree by induction on depth.

□

From now on, we sometimes call the labeled tree of a proposition associated formation tree.

Once we have formation tree of a proposition, we can define these following terms.

Definition 5. *Given an associated formation tree, we have*

1. *The depth of a proposition is the depth of associated formation tree.*
2. *The support of a proposition is the set of propositional letters that occur as labels of the leaves of the associated formation tree.*

Actually, all inductive proof could be based on the depth of a proposition.

4 Parsing

There are infinitely many sequence of symbols. How to identify which is well-defined and which is not? If we want to build a proof system executed on a computer it is very important to construct first a method to check whether it is valid or not.

4.1 Algorithm

As a well-defined propostition can be uniquely mapped into a formation tree, which is easy to check every node well-defined or not. We can now introduce a reacursive algorith to analyze sequence of symbols.

1. If all leaf nodes are labeled with proposition letters, stop it. Otherwise select a leaf node having expressions other than letter and examine it.
2. The first symbol must be (. if the second symbol is $\neg$, jump to step 4. Otherwise go to step 3.
3. (a) Scan the expression from the left until first reaching (α, where α is a nonempty expression having a balance between (and).
   (b) The α is the first of the two constituents.
   (c) The next symbol must be $\wedge, \vee, \rightarrow$, or $\leftrightarrow$.
   (d) The remainder of the expression, β) must consist of a an expression β and).
   (e) Extend the tree by adding α and β as left and right immediate successor respectively.
4. The first two symbols are now known to be ($\neg$. The remainder of the expression, β) must consist of a an expression β and). Then we extend the tree by adding β as its immediate successor. Goto step 1.

In algorithm, we omit the procedure to exit for sequence which is not well-defined. Check every node, if a internal node is not well defined or terminal node is not a proposition letter, the algorithm just stops and asserts a non-well-defined sequence.

What's most important is that the algorithm checks the sequence recursively. It is the similar to the approach to define a well-defined proposition.

4.2 Parentheses

Parentheses are tedious for us to handle. We can introduce some rule to reduce the number of parentheses without ambiguity.

1. The outermost parenthesis need not be explicitly mentioned.
2. The negation symbol applies to as little as possible.
3. The conjuction and disjunction symbols apply to as little as possible.

4. Where one connective symbols is used repeatedly, grouping is to the right.

All these explicit rules make our parsing algorithm complicated if you want to accommodate the right proposition other than well-defined ones for convenience. Furthermore, you should prove that explicit rule would not result in ambiguity.

4.3 Polish notation

Rules are not convenient. However there is a smart way to get rid of parentheses permanently, it is just the Polish notation. As we have already known, any connective is just a function with one or two parameters. We can represent a proposition as following without any parentheses:

1. $\mathcal{D}_\neg(\alpha) = \neg\alpha$.
2. $\mathcal{D}_\vee(\alpha, \beta) = \vee\alpha\beta$.
3. $\mathcal{D}_\wedge(\alpha, \beta) = \wedge\alpha\beta$.
4. $\mathcal{D}_\rightarrow(\alpha, \beta) = \rightarrow \alpha\beta$.
5. $\mathcal{D}_\leftrightarrow(\alpha, \beta) = \leftrightarrow \alpha\beta$.

It is called Polish notation. Actually, it can be expanded to any n-ary function.

Consider the following example, which is translated from a well-defined proposition.

Example 4. *Determine the well defined proposition of*

$$\rightarrow \wedge AD \vee \neg B \leftrightarrow CB.$$

It is easy to recover the well-defined form. Do it as an exercise.

Yah! We do eliminate parentheses successfully. But what is your feeling about Polish notation when you try to figure out what it represents. In fact, it is an approach very suitable for machine but human. To us, the well-defined way is much better.

There are several approached to translate Polish representation into regular one, which is left as an exercise.

5 Appendix: Inductive definition

In our class, many definitions are defined recursively or inductively. We just introduce

Definition 6. *A set S is* closed *under a single operation $f(s_1, \ldots, s_n)$ if and only if every $s_1, \ldots, s_n \in S, f(s_1, \ldots, s_n) \in S$*

Definition 7. *The* closure *of a set S under (all) the operations in a set T is the smallest C such that*

1. $S \subseteq C$ *and*

2. *if* $f \in T$ *is* n*-ary and* $s_1, \ldots, s_n \in C$*, then* $f(s_1, \ldots, s_n) \in C$*.*

With these concepts, we have an Theorem on the property of inductive definition.

Theorem 8. *Suppose that* S *is closed under the operations of* T*, there is a smallest closure* C *such that* $S \subseteq C$*.*

Proof. Given a set D, $S \subseteq D$ and D is closed under the operations of T. Consider the set

$$C = \cap\{D | S \subseteq D \wedge D \text{ is closed under the operations of } T\}.$$

It is obvious that $S \subseteq C$. And C is the smallest set according to definition of C. □

As a direct application of Theorem 8, we have the following Theorem about well-defined Definition:

Theorem 9. *If* S *is a set of well defined propositions and is closed under all five connectives, then* S *is the set of all well defined propositions*

Exercises

1. Show that there are no well-defined propositions of length $2, 3$, or 6, but that any other positive length is possible.

2. Given an example such that $(\alpha \wedge \beta) = (\gamma \wedge \delta)$ but $\alpha \neq \gamma$, where α and β are well-defined and γ and δ are two expressions.

3. Draw formation tree of the following propositions:

   (a) $((A \vee B) \rightarrow ((\neg C) \wedge D))$.
   (b) $(((\neg A) \vee B) \wedge (A \vee (\neg B)))$.

4. Let α be a well-defined proposition; let c be the number of places at which binary connective symbols$(\wedge, \vee, \rightarrow, \leftrightarrow)$ occur in α; let s be the number of places at which proposition letters occur in α. Show by using the induction principle that $s = c + 1$.

5. Determine the Polish notation of proposition or vice versa.

   (a) $((A \vee B) \rightarrow ((\neg C) \wedge D))$.
   (b) $(((\neg A) \vee B) \wedge (A \vee (\neg B)))$.
   (c) $\rightarrow \wedge \neg AB \leftrightarrow C \neg D$.

6. Design a algorithm which can translate Polish notation of proposition into well defined proposition without stack or with at most one.

7. Design a algorithm to translate a well-defined proposition into a Polish notation form.