Discrete Mathematics (II) Spring 2017

Lab Implement Propositional CST of Tableau Proof

Due date: 2017/06/16 24:00

This project is to implement an algorithm for CST in propositional logic. Once you finish this lab, we do think that you should be very familiar with CST.

1 Background

We first figure out 12 atomic tableaux according to truth table as shown in Figure 1.

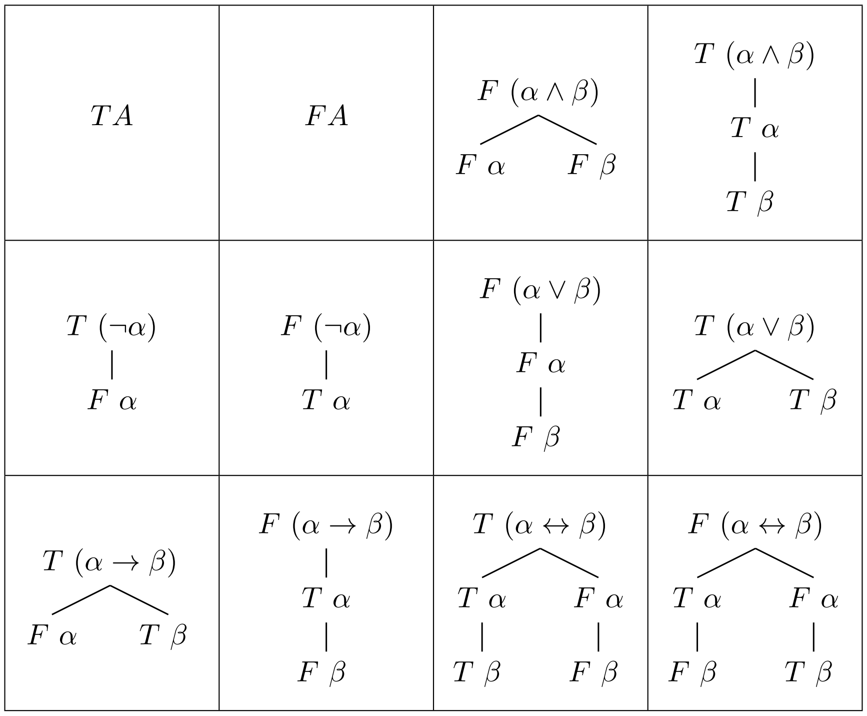

Figure 1: Atomic tableaux of propositional logic

With these atomic tableaux, we can expand a given proposition with a sign following this way:

Definition 1 (Tableaux)**.** *A finite tableau is a binary tree, labeled with signed propositions called entries, such that:*

1. *All atomic tableaux are finite tableaux.*
2. *If τ is a finite tableau, P a path on τ, E an entry of τ occurring on P and ${}'\tau$ is obtained from τ by adjoining the unique atomic tableau with root entry E to τ at the end of the path P, then ${}'\tau$ is also a finite tableau.*

If $\tau_0, \tau_1, \ldots, \tau_n, \ldots$ is a (finite or infinite) sequence of the finite tableaux such that, for each $n \geq 0, \tau_{n+1}$ is constructed from τ_n by an application of (2), then $\tau = \cup \tau_n$ is a tableau.

Remember that a tree τ_i is a partially-ordered set. As τ_i is expanded from τ_{i-1}, we have $\tau_{i-1} \subset \tau_i$.

In order to characterize a tableaux, we define the following terms:

Definition 2. *Let τ be a tableau, P a path on τ and E an entry occurring on P.*

1. *E has been reduced on P if all the entries on one path through the atomic tableau with root E occur on P.*
2. *P is contradictory if, for some proposition α, $T\alpha$ and $F\alpha$ are both entries on P. P is finished if it is contradictory or every entry on P is reduced on P.*
3. *τ is finished if every path through τ is finished.*
4. *τ is contradictory if every path through τ is contradictory.*

As a tableau is at most a binary-branching tree and also be a partial ordered set. However, if we can make it order we can reduce node by node without omitting. Then we just assure a lexical graphical order, which is defined level by level and left to right on a given tableau. We just call this approach Complete Systematic Tableaux (CST) as the following.

Definition 3 (Complete systematic tableaux)**.** *Let R be a signed proposition. We define the complete systematic tableau(CST) with root entry R by induction.*

1. *Let τ_0 be the unique atomic tableau with R at its root.*
2. *Assume that τ_m has been defined. Let n be the smallest level of τ_m and let E be the leftmost such entry of level n.*
3. *Let τ_{m+1} be the tableau gotten by adjoining the unique atomic tableau with root E to the end of every noncontradictory path of τ_m on which E is unreduced.*

The union of the sequence τ_m is our desired complete systematic tableau.

2 Tableau proof

In this section, we design an algorithm to determine whether a given proposition is valid or not. If it is not valid, we hope it can show us an counterexample.

2.1 Algorithm

As we know, the *CST* of a specific Σ and α leads to a unique tree. So we can build this tree step by step. The following is the rough algorithm to make a tableau proof/refutation without premises.

1. Read a line from input file. The root of tableau is Fα.

2. Fetch the next entry not yet processed with smallest number, say Fα. Check whether it is reduced or not.
   (a) If it is reduced, e.g. the root of atomic tableau and a propositional letter, just ignore it.
   (b) If it is unreduced. Check whether Tα occurs along the path. If it does, close this path for contradiction. Otherwise, reduce it and adjoin its corresponding atomic tableau into every noncontradictory path through it.

In this algorithm, you should clearly determine the order of an entry in tableau. We have discussed this question in exercise. It is important to remember that the root entry has the order 0, the smallest one. One thing is sure that newly generated entries are never smaller than currently being reduced entry.

2.2 Counterexample

Once the CST is built, you can determine whether the tableau is contradictory or not. As Definition 5 shown, if Tα and Fα occurs on a path. We just omit it. If there are some noncontradictory paths. We can choose one and construct a counterexample to make α false.

The counterexample can be constructed by searching along the path. When T `A_{1}` is met, we just put `A_{1}` into a set, which consists of propositional letters which are assigned true. The other propositional letters are assigned false whether they occurs on the path or not.

The point is to backtrack a noncontradictory path to construct a counterexample. So you need design a proper data structure to represent tableau.

2.3 Input and output

In this section, we define the format for input and output. So it is easy to feed your implementation and verify its correctness. Because CST with premises is a determined procedure with a specific input. It means that the output should be always the same.

2.3.1 Input

The program take as input a set of propositions in a text file. Each line is a proposition α to check. When you reach the end of input file, you can safely terminate your program.

The following is an example of input.

```
(A_{1} \or B_{1})
(A_{1} \imply B_{2})
(A_{2} \and B_{1})
```

2.3.2 Output

The out put is also a formated text file. When an entry is being reduced, you just output it. **So the output just records the sequence you reducing the tableau**.

$$\begin{tikzcd}
\text{F } (A_1 \vee B_1) \arrow[d, dash] \\
\text{F } (A_1 \vee B_1) \arrow[d, dash] \\
\text{F } A_1 \arrow[d, dash] \\
\text{F } B_1
\end{tikzcd}$$

Figure 2: The tableau proof of $\vdash (A_1 \vee B_1)$

With the input in Section 2.3.1, we have the following output, here we only consider the first line of input.

```
F (A_{1} \or B_{1})
F (A_{1} \or B_{1})
F A_{1}
F B_{1}
```

You can construct the tableau proof and verify that this output is just the sequence to reduce entries in its corresponding CST.

Actually, the tableau proof is shown as Figure 2.

If a tableau is not contradictory. You put all proposition letters which are assigned with truth value T immediately following the line of the last reduced entry beginning with "counterexample", e.g. A_{1} A_{2}.

Here, we just show you an input and its corresponding output. Given an input:

```
((A_{1} \or B_{1}) \and (A_{1} \imply B_{2}))
```

Its corresponding out put is as following:

```
F ((A_{1} \or B_{1}) \and (A_{1} \imply B_{2}))
F ((A_{1} \or B_{1}) \and (A_{1} \imply B_{2}))
F (A_{1} \or B_{1})
F (A_{1} \imply B_{2})
F (A_{1} \or B_{1})
F (A_{1} \imply B_{2})
F A_{1}
T A_{1}
F B_{1
F B_{2}
counterexample
A_{1}
```

The CST is shown as Figure 3. The tableau is not contradictory. The output just show a counterexample constructed along the rightest path.

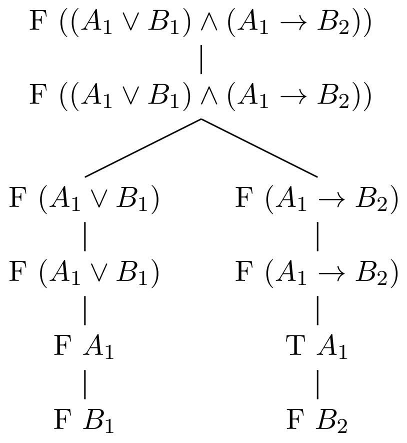

Figure 3: The tableau proof of $((A_1 \vee B_1) \wedge (A_1 \leftarrow B_2))$

3 Parsing algorithm of proposition

First of all, the program should determine whether an input is valid or not. In this section, we define how to represent a well-defined proposition for program.

In this Lab, we just focus on how to construct a complete systematic tableau proof of propositional logic. So we needn't implement parsing algorithm very robust like a compiler except that it recognize only well defined propositions. In another word, proposition letter and connectives in given proposition strictly follow the form defined in this document.

3.1 Representation of proposition

Some symbols are defined in propositional logic. We call it *language*, all complicated strings are constructed based on these symbols with some rules.

Definition 4. *The* language *of propositional logic consists of the following symbols:*

1. *Connectives:* $\vee, \wedge, \neg, \to, \leftrightarrow$
2. *Parentheses:), (*
3. *Propositional Letters:* $A, A_1, A_2, \cdots, B, B_1, B_2, \cdots$.

where we suppose the set of proposition letters is countable and a proposition letter can only be a capital letter with some subscript.

An inductive approach is defined to construct a proposition.

Definition 5 (Proposition)**.** *A* proposition *is a sequence of many symbols which can be constructed in the following approach:*

1. *Propositional letters are propositions.*
2. *if α and β are propositions, then $(\alpha \vee \beta), (\alpha \wedge \beta), (\neg\alpha), (\alpha \to \beta)$ and $(\alpha \leftrightarrow \beta)$ are propositions.*

connective	notation
$\wedge$	`\and`
$\vee$	`\or`
$\neg$	`\not`
$\rightarrow$	`\imply`
$\leftrightarrow$	`\eq`

Table 1: Notation of propositions

3. *A string of symbols is a proposition if and only if it can be obtained by starting with propositional letters (1) and repeatedly applying (2).*

It is obvious that infinite propositions can be generated even if there are only finite proposition letters.

A proposition like $((A_1 \wedge A_2) \rightarrow B)$ is hard to represent with ascii character because of subscripts and connectives. Here we define some notations for propositions.

Definition 6. *Every connective begins with a special character* `\` *as shown in Table* **??**. *And proposition letter A_{ij} is represented as* `A_{ij}`*.*

Furthermore, the alphabet of proposition consist of only capitals. The subscript is just a natural number. In Table 1, subscript ij can be any letters. But We just let them be a concrete number for simplcity, e.g. natural number.

According to Definition 6, the proposition $((A_1 \wedge A_2) \rightarrow B)$ is a symbol sequence as
`((A_{1} \and A_{2}) \imply B).`

3.2 Parsing algorithm

As a well-defined proposition can be uniquely mapped into a formation tree, which is easy to check every node well-defined or not. We can now introduce a recursive algorithm to analyze a 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, $\beta)$ 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.

Given a sequence of symbols in Definition 6, the parsing algorithm should be deliberately modified to accommodate those representations. Furthermore, the depth of an input proposition is unknown in advance, the buffer should be allocated dynamically.

And you should be aware that the formation tree of a proposition is 2-branch and the length of proposition is not fixed in a node.

4 Requirement

Given a proposition α, you need to implement the process of the CST of the α. You should show how you process the tableau proof of input, give us every entry you reduced in a right order and what the entry is. You just need to show the process with a string which is connection of every entry and which entry is according to the format we required. We will only test whether the result string of your program is the same as the standard answer we provide. So you must pay attention to the output format.

Hint: You should do the CST as standard step of definition, which means you can't omit any entry, though this step is just copy that entry.

5 Development environment

You can finish the project in language you wished. But we recommend Java and C. The Linux distribution Ubuntu 16.04 is a good choice, which is easy to install and use. It also provides GCC, Java, and related environment. For simplicity, you can install Ubuntu in a virtual machine. VirtualBox is free to download and works on Windows, Linux, and Mac OS.

6 Submission and grading

Finally, you should submit a package of you source code and a document describing you design. You should submit all belonging to school's ftp site.

Whatever language is used, the document should describes how to build and deploy your program. If TA cannot run your program with your instruction, you will fail this project. You running program take as input a file. You program just output a file as requirement. You should construct a set of propositions to verify your implementation.

A benchmark is used to grade your implementation. It contains a little statement . We will try to cover all cases.Some complicate propositions are also generated to test an input with arbitrary length.

7 Help

Finally, you can contact us immediately if you have some question on the project. Please send email to liy@fudan.edu.cn.