Fall 2010 CIS 160

Mathematical Foundations of Computer Science
Jean Gallier

Mini Project

October 18, 2010; Due December 9, 2010
This project must be done **individually**

This project consists of a blend of theory and computer implementation. You are only required to give answers to Parts 1 and 2. These constitute 10% of your total grade. Part 3, the programming part is *optional* and will count only for extra credit (up to 10% of your total grade).

If you decide do do part 3, the code implementing the various required algorithms may be written in Java, C++, Python, Mathematica or Matlab. Please submit the source for every program in addition to every required output. Please pay attention to the readability of every output (and make sure your code is suitably documented). You may submit the various requested proofs in a separate document.

Even though natural deduction proof systems for classical propositional logic are complete (with respect to the truth values semantics), they are not adequate for designing algorithms searching for proofs (because of the amount of nondeterminism involved).

Gentzen designed a different kind of proof system using *sequents* (later refined by Kleene, Smullyan and others) that is far better suited for the design of automated theorem provers. Using such a proof system (a *sequent calculus*), it is relatively easy to design a procedure that terminates for all input propositions, P, and either certifies that P is (classically) valid or else returns some (or all) falsifying truth assignment(s) for P. In fact, if P is valid, the tree returned by the algorithm can be viewed as a proof of P in this proof system.

For this mini-project, we describe a *Gentzen sequent-calculus*, G', for propositional logic that lends itself well to the implementation of algorithms searching for proofs or falsifying truth assignments of propositions.

Such algorithms build trees whose nodes are labeled with pairs of sets called sequents. A *sequent* is a pair of sets of propositions denoted by

$$P_1, \ldots, P_m \to Q_1, \ldots, Q_n,$$

with $m, n \geq 0$. Symbolically, a sequent is usally denoted $\Gamma \to \Delta$, where Γ and Δ are two finite sets of propositions (not necessarily disjoint).

For example,

$$\to P \Rightarrow (Q \Rightarrow P), \quad P \vee Q \to, \quad P, Q \to P \wedge Q$$

are sequents. The sequent $\to$, where both $\Gamma = \Delta = \emptyset$ corresponds to falsity.

The choice of the symbol, $\to$, to separate the two sets of propositions Γ and Δ is commonly used and was introduced by Gentzen but there is nothing special about it. If you don't like it, you may replace it by any symbol of your choice as long as that symbol does not clash with the logical connectives $(\Rightarrow, \wedge, \vee, \neg)$. For example, you could denote a sequent

$$P_1, \ldots, P_m; Q_1, \ldots, Q_n,$$

using the semicolon as a separator.

Given a truth assignment, v, to the propositional letters in the propositions P_i and Q_j, we say that v *satisfies the sequent*, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, iff

$$v((P_1 \wedge \cdots \wedge P_m) \Rightarrow (Q_1 \vee \cdots \vee Q_n)) = \mathbf{true},$$

or equivalently, v *falsifies the sequent*, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, iff

$$v(P_1 \wedge \cdots \wedge P_m \wedge \neg Q_1 \wedge \cdots \wedge \neg Q_n) = \mathbf{true},$$

iff

$$v(P_i) = \mathbf{true},\ 1 \leq i \leq m \quad \text{and} \quad v(Q_j) = \mathbf{false},\ 1 \leq j \leq n.$$

A sequent is *valid* iff it is satisfied by all truth assignments iff it cannot be falsified.

Note that a sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, can be falsified iff some truth assignment satisfies all of $P_1, \ldots, P_m$ and falsifies all of $Q_1, \ldots, Q_n$. In particular, if $\{P_1, \ldots, P_m\}$ and $\{Q_1, \ldots, Q_n\}$ have some common proposition (they have a nonempty intersection), then the sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, is valid. On the other hand if all the P_i's and Q_j's are propositional letters and $\{P_1, \ldots, P_m\}$ and $\{Q_1, \ldots, Q_n\}$ are disjoint (they have no symbol in common), then the sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, is falsified by the truth assignment, v, where $v(P_i) = \mathbf{true}$, for $i = 1, \ldots m$ and $v(Q_j) = \mathbf{false}$, for $j = 1, \ldots, n$.

In the special case where $m = 0$, the truth assignment v falsifies the sequent $\to Q_1, \ldots, Q_n$ iff $v(Q_j) = \mathbf{false}$ for $j = 1, \ldots, n$, and in the special case where $n = 0$, the truth assignment v falsifies the sequent $P_1, \ldots, P_m \to$ iff $v(P_i) = \mathbf{true}$ for $i = 1, \ldots m$. Thus, an empty left-hand side of a sequent is interpreted as **true** and an empty right-hand side of a sequent is interpreted as **false**. In particular, the degenerate sequent $\to$ is falsified by all truth assignments.

The main idea behind the design of the proof system G' is to systematically *try to falsify a sequent*. If such an attempt fails, the sequent is valid and a proof tree is found. Otherwise, all falsifying truth assignments are returned. In some sense

failure to falsify is success (in finding a proof)!

The rules of G' are designed so that the conclusion of a rule is falsified by a truth assignment, v, iff its single premise of one of its two premises is falsified by v. Thus, these rules can be viewed as *two-way* rules that can either be read bottom-up or top-down.

Here are the axioms and the rules of the *sequent calculus* G':

Axioms: $\Gamma, P \to P, \Delta$

Inference rules:

$$\frac{\Gamma, P, Q, \Delta \to \Lambda}{\Gamma, P \wedge Q, \Delta \to \Lambda} \ \wedge\text{: left} \qquad \frac{\Gamma \to \Delta, P, \Lambda \quad \Gamma \to \Delta, Q, \Lambda}{\Gamma \to \Delta, P \wedge Q, \Lambda} \ \wedge\text{: right}$$

$$\frac{\Gamma, P, \Delta \to \Lambda \quad \Gamma, Q, \Delta \to \Lambda}{\Gamma, P \vee Q, \Delta \to \Lambda} \ \vee\text{: left} \qquad \frac{\Gamma \to \Delta, P, Q, \Lambda}{\Gamma \to \Delta, P \vee Q, \Lambda} \ \vee\text{: right}$$

$$\frac{\Gamma, \Delta \to P, \Lambda \quad Q, \Gamma, \Delta \to \Lambda}{\Gamma, P \Rightarrow Q, \Delta \to \Lambda} \ \Rightarrow\text{: left} \qquad \frac{P, \Gamma \to Q, \Delta, \Lambda}{\Gamma \to \Delta, P \Rightarrow Q, \Lambda} \ \Rightarrow\text{: right}$$

$$\frac{\Gamma, \Delta \to P, \Lambda}{\Gamma, \neg P, \Delta \to \Lambda} \ \neg\text{: left} \qquad \frac{P, \Gamma \to \Delta, \Lambda}{\Gamma \to \Delta, \neg P, \Lambda} \ \neg\text{: right}$$

where Γ, Δ, Λ are any finite sets of propositions, possibly the empty set.

A *deduction tree* is either a one-node tree labeled with a sequent or a tree constructed according to the rules of system G'. A *proof tree* (or *proof*) is a deduction tree whose leaves are *all* axioms. A proof tree for a proposition, P, is a proof tree for the sequent, $\to P$ (with an empty left-hand side).

For example,

$$P, Q \to P$$

is a proof tree.

Here is a proof tree for $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{P, \neg Q \to P}{\neg Q \to \neg P, P}}{\to P, (\neg Q \Rightarrow \neg P)} \qquad \dfrac{\dfrac{Q \to Q, \neg P}{\neg Q, Q \to \neg P}}{Q \to (\neg Q \Rightarrow \neg P)}}{(P \Rightarrow Q) \to (\neg Q \Rightarrow \neg P)}}{\to (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)}$$

The following is a deduction tree but not a proof tree:

$$\dfrac{\dfrac{\dfrac{P, R \to P}{R \to \neg P, P}}{\to P, (R \Rightarrow \neg P)} \qquad \dfrac{\dfrac{R, Q, P \to}{R, Q \to \neg P}}{Q \to (R \Rightarrow \neg P)}}{\dfrac{(P \Rightarrow Q) \to (R \Rightarrow \neg P)}{\to (P \Rightarrow Q) \Rightarrow (R \Rightarrow \neg P)}}$$

since its rightmost leaf, $R, Q, P \to$, is falsified by the truth assignment $v(P) = v(Q) = v(R) = \mathbf{true}$, which also falsifies $(P \Rightarrow Q) \Rightarrow (R \Rightarrow \neg P)$.

Let us call a sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, *finished* if either it is axiom ($P_i = Q_j$ for some i and some j) or all the propositions P_i and Q_j are atomic and $\{P_1, \ldots, P_m\} \cap \{Q_1, \ldots, Q_n\} = \emptyset$. We will also say that a deduction tree is finished if all its leaves are finished sequents.

The beauty of the proof system G' is that for every sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, the process of building a deduction tree from this sequent *always terminates with a tree where all leaves are finished, independently of the order in which the rules are applied*. Therefore, we can apply any strategy we want when we build a deduction tree and we are sure that we will get a deduction tree with all its leaves finished. If all the leaves are axioms, then we have a proof tree and the sequent is valid, or else all the leaves that are not axioms yield a falsifying assignment, and all falsifying assignments for the root sequent are found this way.

If we only want to know whether a proposition (or a sequent) is valid, we can stop as soon as we find a finished sequent that is not an axiom since in this case, the input sequent is falsifiable.

(1) (**Required**) Prove that for every sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, any sequence of applications of the rules of G' terminates with a deduction tree whose leaves are all finished sequents (a finished deduction tree).

Hint. Define the number of connectives, $c(P)$, in a proposition, P, as follows:

(1) If P is a propositional symbol, then

$$c(P) = 0.$$

(2) If $P = \neg Q$, then

$$c(\neg Q) = c(Q) + 1.$$

(3) If $P = Q * R$, where $* \in \{\Rightarrow, \vee, \wedge\}$, then

$$c(Q * R) = c(Q) + c(R) + 1.$$

Given a sequent,

$$\Gamma \to \Delta = P_1, \ldots, P_m \to Q_1, \ldots, Q_n,$$

define the number of connectives, $c(\Gamma \to \Delta)$, in $\Gamma \to \Delta$ by

$$c(\Gamma \to \Delta) = c(P_1) + \cdots + c(P_m) + c(Q_1) + \cdots + c(Q_n).$$

Prove that the application of every rule decreases the number of connectives in the premise(s) of the rule.

(2) (**Required**) Prove that for every sequent, $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$, for every finished deduction tree, T, constructed from $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$ using the rules of G', every truth assignment, v, satisfies $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$ iff v satisfies every leaf of T. Equivalently, a truth assignment, v, falsifies $P_1, \ldots, P_m \to Q_1, \ldots, Q_n$ iff v falsifies some leaf of T.

Deduce from the above that a sequent is valid iff all leaves of every finished deduction tree, T, are axioms. Furthermore, if a sequent is not valid, then for every finished deduction tree, T, for that sequent, every falsifying assignment for that sequent is a falsifying assignment of some leaf of the tree, T.

(3) **Programming Project; Extra Credit**:

Design an algorithm taking any sequent as input and constructing a finished deduction tree. If the deduction tree is a proof tree, output this proof tree in some fashion (such a tree can be quite big so you may have to find ways of "flattening" these trees). If the sequent is falsifiable, stop when the algorithm encounters the first leaf which is not an axiom and output the corresponding falsifying truth assignment.

I suggest using a *depth-first expansion strategy* for constructing a deduction tree. What this means is that when building a deduction tree, the algorithm will proceed recursively as follows: Given a non-finished sequent

$$A_1, \ldots, A_p \to B_1, \ldots, B_q,$$

if A_i is the *leftmost* non-atomic proposition if such proposition occurs on the left or if B_j is the leftmost non-atomic proposition if all the A_i's are atomic, then

(1) The sequent is of the form

$$\Gamma, A_i, \Delta \to \Lambda,$$

with A_i the leftmost non-atomic proposition. Then either

(a) $A_i = C_i \wedge D_i$ or $A_i = \neg C_i$, in which case either we recursively construct a (finished) deduction tree

$$\begin{array}{c} \mathcal{D}_1 \\ \Gamma, C_i, D_i, \Delta \to \Lambda \end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma, C_i, D_i, \Delta \to \Lambda\end{array}}{\Gamma, C_i \wedge D_i, \Delta \to \Lambda}$$

or we recursively construct a (finished) deduction tree

$$\begin{array}{c}\mathcal{D}_1\\ \Gamma, \Delta \to C_i, \Lambda\end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma, \Delta \to C_i, \Lambda\end{array}}{\Gamma, \neg C_i, \Delta \to \Lambda}$$

or

(b) $A_i = C_i \vee D_i$ or $A_i = C_i \Rightarrow D_i$, in which case either we recursively construct two (finished) deduction trees

$$\begin{array}{c}\mathcal{D}_1\\ \Gamma, C_i, \Delta \to \Lambda\end{array} \qquad \text{and} \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma, D_i, \Delta \to \Lambda\end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma, C_i, \Delta \to \Lambda\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma, D_i, \Delta \to \Lambda\end{array}}{\Gamma, C_i \vee D_i, \Delta \to \Lambda}$$

or we recursively construct two (finished) deduction trees

$$\begin{array}{c}\mathcal{D}_1\\ \Gamma, \Delta \to C_i, \Lambda\end{array} \qquad \text{and} \qquad \begin{array}{c}\mathcal{D}_2\\ D_i, \Gamma, \Delta \to \Lambda\end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma, \Delta \to C_i, \Lambda\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ D_i, \Gamma, \Delta \to \Lambda\end{array}}{\Gamma, C_i \Rightarrow D_i, \Delta \to \Lambda}$$

(2) The non-finished sequent is of the form

$$\Gamma \to \Delta, B_j, \Lambda,$$

with B_j the leftmost non-atomic proposition. Then either

(a) $B_j = C_j \vee D_j$ or $B_j = C_j \Rightarrow D_j$, or $B_j = \neg C_j$, in which case either we recursively construct a (finished) deduction tree

$$\begin{array}{c} \mathcal{D}_1 \\ \Gamma \to \Delta, C_j, D_j, \Lambda \end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c} \mathcal{D}_1 \\ \Gamma \to \Delta, C_j, D_j, \Lambda \end{array}}{\Gamma \to \Delta, C_j \vee D_j, \Lambda}$$

or we recursively construct a (finished) deduction tree

$$\begin{array}{c} \mathcal{D}_1 \\ C_j, \Gamma \to D_j, \Delta, \Lambda \end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c} \mathcal{D}_1 \\ C_j, \Gamma \to D_j, \Delta, \Lambda \end{array}}{\Gamma \to \Delta, C_j \Rightarrow D_j, \Lambda}$$

or we recursively construct a (finished) deduction tree

$$\begin{array}{c} \mathcal{D}_1 \\ C_j, \Gamma \to \Delta, \Lambda \end{array}$$

to get the deduction tree

$$\dfrac{\begin{array}{c} \mathcal{D}_1 \\ C_j, \Gamma \to \Delta, \Lambda \end{array}}{\Gamma \to \Delta, \neg C_j, \Lambda}$$

or

(b) $B_j = C_j \wedge D_j$, in which case we recursively construct two (finished) deduction trees

$$\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \to \Delta, C_j, \Lambda & \Gamma \to \Delta, D_j, \Lambda \end{array} \quad \text{and}$$

to get the deduction tree

$$\dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \to \Delta, C_j, \Lambda & \Gamma \to \Delta, D_j, \Lambda \end{array}}{\Gamma \to \Delta, C_j \wedge D_j, \Lambda}$$

If you prefer, you can apply a *breadth-first expansion strategy* for constructing a deduction tree.

Input Format.

In order to make life easier, I suggest that the input propositions be given in postfix notation because it is easy to build a tree from an expression in postfix, using a stack.

Also, use "&" for $\wedge$, "|" for $\vee$, "$\sim$" for $\neg$ and ">" for $\Rightarrow$ (or any other reasonable choice of characters).

Each propositional letter $\mathbf{P}_i$ is represented by the string "Pi". For example, $\mathbf{P}_{17}$ will be represented by P17.

Use spaces as delimiters.

The postfix representation is specified recursively as follows:

1. $\text{postfix}(P_i) = Pi$
2. $\text{postfix}(P \vee Q) = \text{postfix}(P)\ \text{postfix}(Q)\ |$
3. $\text{postfix}(P \wedge Q) = \text{postfix}(P)\ \text{postfix}(Q)\ \&$
4. $\text{postfix}(P \Rightarrow Q) = \text{postfix}(P)\ \text{postfix}(Q)\ >$
5. $\text{postfix}(\neg P) = \text{postfix}(P)\ \sim$

For example, the proposition

$$(\mathbf{P}_1 \vee \neg \mathbf{P}_2) \wedge (\neg \mathbf{P}_2 \Rightarrow \mathbf{P}_3)$$

has the following postfix representation:

$$P1\ P2\ \sim |\ \ P2\ \sim\ P3\ >\ \&$$

Output Format.

Your program should output a Gentzen-deduction tree in legible form and specify whether this tree is a proof tree or not. The nodes of deduction trees are Gentzen sequents, that is, expressions of the form

$$P_1, \ldots, P_m \ => Q_1, \ldots, Q_n,$$

where the P_is and Q_is are propositions in postfix notation (or if you work harder, in standard infix notation), separated by some delimiter such as =>.