PC Trees and PC Arithmetic

CS 4710 Course Notes

January 29, 2018

Propositional Calculus Trees (PCT)

Consider the problem of searching for counterexamples to an argument. We can perform an uninformed search by drawing the entire truth table and inspecting it brute-force.

Example. Consider the simple argument $P, P \to Q \vdash Q$.

P	$P \to Q$	Q
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\top$
$\bot$	$\top$	$\bot$

By inspecting the above we can see the argument is valid (no counterexamples exist), as counterexamples to this argument only exist when $[\![P]\!] = \top, [\![P \to Q]\!] = \top$, and $[\![Q]\!] = \bot$. By inspection, there is no such row in the truth table. However, we can already see a short cut of sorts is possible – to check for a counterexample, we really only need to check the valuations where the premises turn out true along with the conclusion turning out false.

PC Trees (PCT as we'll call it) give us some effective procedures for taking these short cuts. In order to use them, however, we will have to establish that they are a proper representation of PC.

PCT for Counterexamples

First we'll see how to use the construction of PCT to detect counterexamples in an argument of PC.

Example. Consider the argument $A \vee B, \neg A \vdash B$. To construct our tree, we:

(1) List our premises vertically (as a single "branch") and **negate** the conclusion.

$$\begin{gathered} A \vee B \\ \neg A \\ \neg B \end{gathered}$$

Notice that the above set of statements are satisfiable if and only if there exists a counterexample to our argument. In other words, if the set containing the premises of our original argument and the negation of the conclusion is satisfiable, then there exists a counterexample to the argument.

Therefore, it is not valid.

(2) “Tick” compound statements and expand the tree using our tree rules. In this example we’re using the “disjunctive” rule in which each disjunct gets its own branch. We’ll introduce the tree rules in entirety below.

$$\begin{tikzcd}[row sep=small, column sep=small]
& A \lor B \checkmark & \\
& \neg A & \\
& \neg B \arrow[dl, no head] \arrow[dr, no head] & \\
\mathrm{A} & & \mathrm{B}
\end{tikzcd}$$

We only have one compound statement ($A \lor B$) in our argument. As we’ll see below, a non-negated disjunction is split into branches. Now we check our generated paths to see if they are **closed**. If they aren’t closed, they are **open**. A closed path is one in which our generated statement is exactly negated by a statement earlier on the same path. (One can think of this as generating the cases for a proof by contradiction, which is very similar in flavor.)

Both A and B are closed in our example, as $\neg A$ and $\neg B$ both exist on the branch above. In other words, if we trace upwards from both branches, we find each negation. Therefore, both branches are closed.

(3) Check if generated paths are closed. If they are open and no longer analyzable, we halt and conclude the path is open. (If a path is closed, we write X’s by the terminus. If it’s open and further unanalyzable, we write a O.)

$$\begin{tikzcd}[row sep=small, column sep=small]
& A \lor B \checkmark & \\
& \neg A & \\
& \neg B \arrow[dl, no head] \arrow[dr, no head] & \\
\mathrm{A}\mathbf{X} & & \mathrm{B}\mathbf{X}
\end{tikzcd}$$

(4) If the tree we generate is fully closed (all paths are closed), then the associated argument is valid. This is our test function.

As we can see, $A \lor B, \neg A \vdash B$ is valid.

Example. Now let’s take a look at an example of an invalid argument. By the above procedure, if the tree we generate isn’t fully closed, the argument is not valid. Consider the argument $A \lor B, A \vdash B$.

$$\begin{tikzcd}[row sep=small, column sep=small]
& A \lor B \checkmark & \\
& A & \\
& \neg B \arrow[dl, no head] \arrow[dr, no head] & \\
\mathrm{A}\mathbf{O} & & \mathrm{B}\mathbf{X}
\end{tikzcd}$$

Note that our left hand path, in this example, is **open**. This is because we cannot find $\neg A$ above A (in fact, we find A itself). Thus, the argument is invalid.

PCT Rules

The following are rules for “expanding” or “analyzing” compound statements in PCT. Above we rewrote the argument with a negated conclusion in order to find counterexamples (check validity). We will see that we can check for various properties (e.g. tautology, contradiction) by rewriting the original argument in various ways before constructing our full tree. These tree construction rules, however, stay the same no matter which procedure (checking for tautology, validity, etc) we use. (As an exercise, consider what sort of demonstration would be necessary to prove these rules are correct given PC.)

Rule D (Disjunction)

$$\begin{array}{c} P \lor Q\checkmark \\ \diagup \quad \diagdown \\ \mathrm{P} \qquad \mathrm{Q} \end{array}$$

Rule ND (Negated Disjunction)

$$\begin{array}{c} \neg(P \lor Q)\checkmark \\ \neg P \\ \neg Q \end{array}$$

Rule C (Conjunction)

$$\begin{array}{c} P \land Q\checkmark \\ P \\ Q \end{array}$$

Rule NC (Negated Conjunction)

$$\begin{array}{c} \neg(P \land Q)\checkmark \\ \diagup \quad \diagdown \\ \neg P \qquad \neg Q \end{array}$$

Rule I (Implication)

$$\begin{array}{c} P \rightarrow Q\checkmark \\ \diagup \quad \diagdown \\ \neg P \qquad Q \end{array}$$

Rule NI (Negated Implication)

$$\begin{array}{c} \neg(P \rightarrow Q)\checkmark \\ P \\ \neg Q \end{array}$$

Rule DN (Double Negation)

$$\begin{array}{c} \neg\neg P\checkmark \\ P \end{array}$$

Rule N1 (Negation 1)

$$\begin{array}{c} P \\ \neg P \ \mathbf{X} \end{array}$$

Rule N2 (Negation 2)

$$\begin{array}{c} \neg P \\ P \ \mathbf{X} \end{array}$$

In our construction rules, we include rules for closing of paths. If you cannot apply any rules on an open (non-closed) path and it fails the check for being closed, it is marked open.

Correctness and Adequacy of PCT Rules

Define a method as a combination of a representation and an algorithm. For PCT that is the combination of our tree structures along with our tree rules, test function for argument validity, etc. We have two major issues at play:

(1) Does my method actually represent my problem adequately?

(2) What properties does my method have?

Example. We will produce a PCT test for tautology below.

Consider $S \equiv (A \wedge B) \vee (\neg A \vee \neg B)$. Show $\vDash S$ using a PCT based algorithm. Call this the "PCT tautology test". Instead of listing the premises of our argument and negating the conclusion, we will negate **all** of the statements considered (in this case, one statement). We generate a tree from the negated premise, and if each path is closed, we are guaranteed the premise is a tautology. (Consider why this works out.)

Step 1 (use of ND):

$$\begin{array}{c} \neg[(A \wedge B) \vee (\neg A \vee \neg B)]\checkmark \\ \neg(\neg A \vee \neg B) \\ \neg(A \wedge B) \end{array}$$

Step 2 (another use of ND):

$$\begin{array}{c} \neg[(A \wedge B) \vee (\neg A \vee \neg B)]\checkmark \\ \neg(\neg A \vee \neg B) \\ \neg(A \wedge B)\checkmark \\ \neg\neg A \\ \neg\neg B \end{array}$$

Step 3 (use of NC):

$$\begin{array}{c}
\neg[(A \wedge B) \vee (\neg A \vee \neg B)]\checkmark \\
\neg(\neg A \vee \neg B)\checkmark \\
\neg(A \wedge B)\checkmark \\
\neg\neg A \\
\neg\neg B \\
\diagup \quad \diagdown \\
\neg A \qquad \neg B
\end{array}$$

Steps 4, 5 (two uses of N2):

$$\begin{array}{c}
\neg[(A \wedge B) \vee (\neg A \vee \neg B)]\checkmark \\
\neg(\neg A \vee \neg B)\checkmark \\
\neg(A \wedge B)\checkmark \\
\neg\neg A \\
\neg\neg B \\
\diagup \quad \diagdown \\
\neg A\ \mathbf{X} \qquad \neg B\ \mathbf{X}
\end{array}$$

Since all paths are closed, we conclude $\vDash S$.

Definition. Consider a tree of the form:

$$\begin{array}{c}
P\checkmark \\
Q \\
R
\end{array}$$

P is called the **premise**, and $\{Q, R\}$ is the **list of conclusions**.

(Note: any set of statements in PC can be rewritten in either "conjunctive normal form" or "disjunctive normal form", due to the expressive completeness of negation and conjunction, and the expressive completeness of negation and disjunction. We won't cover CNF and DNF explicitly in this course, but you should familiarize yourself with it as it can make computation much easier.)

Definition. Similarly, for the tree form:

$$\begin{array}{c}
P\checkmark \\
\diagup \quad \diagdown \\
\begin{array}{cc}
\text{Q} & \text{Q'} \\
\mid & \mid \\
\text{R} & \text{R'}
\end{array}
\end{array}$$

P is called the premise, and $\{Q, R\}$, $\{Q', R'\}$ are called the lists of conclusions. Each is a list of conclusions.

Definition. Tree rule r is **correct** iff:

$$[\![P]\!]^V = \top \Rightarrow [\![s]\!]^V = \top \text{ for } s \in L\text{, where } L \text{ is one of } r\text{'s lists of conclusions.}$$

Definition. Tree rule r is **complete** iff the premise of r is true whenever $[\![s]\!]^V = \top$ for all $s \in L$, for some list of conclusions L for r.

Note: we're guaranteed to halt when we use these rules. Consider why. Also, note that if an algorithm is guaranteed to halt, that this implies there's some finite worst case for its runtime.

Theorems: Correctness, Completeness, and Adequacy

Theorem (correctness of tree rules): If a finite set S of statements is satisfiable, there will be an open path through any true that starts with S.

Sketch of proof. First note: if all statements in a path ψ of a tree are true under valuation V, then ψ is open. Formally, if for all $s \in \psi : [\![s]\!]^V = \top$, then ψ is open.

If $s, \neg s \in \psi$, then one must be $\bot$ under V by the definition of assignment. However, this violates our assumption.

Suppose for some V, $[\![s]\!]^V = \top$ for $s \in S$. Consider the following property of a tree τ:

(*) τ starts with S and contains a path ψ such that all statements occupying nodes of ψ are true under V.

Claim: if τ has property (*), so does any tree τ^* obtained from applying some tree rule r to τ. Proving this claim will get us to our goal. Note that we are relying on proofs of the correctness of tree rules (which are left as exercises).

The proof of this claim is left as an exercise as well.

Theorem. If τ is associated with argument A and is closed, then A is valid.

Theorem (adequacy of the tree rules): If there is an open path through a finished tree τ starting with a set of statements S, then S is satisfiable.

Sketch of proof. Let τ be a finished tree starting with S and containing an open path ψ. Let V be the valuation under which atomic statements in ψ are true, and all others false.

Claim: all statements in ψ are true under V. (Note this claim implies adequacy immediately.)

To prove the claim, consider the lengths of statements as cases. There are three interesting cases as a result, where the length is 1 (atomic statements), the length is 2 (negated atomic statements), and the length is greater than or equal to 3 (compound statements with binary operators). These proofs are left as exercises. To sketch the proof for the third case, however: suppose some statement P in ψ is false under V. Furthermore, suppose P is the shortest such statement, and go for a proof by contradiction.

Theorem (argument adequacy): If an argument is valid, any finished tree associated with it is closed.

Arithmetic Representation of PC

We will now present a different way of representing PC – using arithmetic constants and operators. Doing so will reveal the algebraic connections between arithmetic, set theoretic, and logical operators that we've seen so far.

Define two constants: $\{0, 1\}$ and two operators $\{+, \cdot\}$ with the following constraints:

(1) $0 + 0 = 1 + 1 = 0$

(2) $0 + 1 = 1 + 0 = 1$

(3) $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$

(4) $1 \cdot 1 = 1$

Let 0 represent $\bot$, 1 represent $\top$, $\cdot$ represent $\wedge$, and $1 + \ldots$ represent $\neg$. (Here's where a "representation proof" would be nice, assuring us that these representations don't run afoul of anything. Consider how one might be done.)

How would we interpret $\vee$ and $\rightarrow$ in this system? Any proof of representation would have to demonstrate that our operators turn out properly, after all. Applying our definitions we can do some algebra.

For $\vee$:

$$\begin{gathered}
[\![A \vee B]\!] = [\![\neg(\neg A \wedge \neg B)]\!] \\
= 1 + (1 + [\![A]\!]) \cdot (1 + [\![B]\!]) \\
= 1 + 1 + [\![A]\!] + [\![B]\!] + [\![A]\!] \cdot [\![B]\!] \\
= 0 + [\![A]\!] + [\![B]\!] + [\![A]\!] \cdot [\![B]\!] \\
= [\![A]\!] + [\![B]\!] + [\![A]\!] \cdot [\![B]\!]
\end{gathered}$$

For $\rightarrow$:

$$\begin{gathered}
[\![A \rightarrow B]\!] = [\![\neg A \vee B]\!] \\
= 1 + [\![A]\!] + [\![B]\!] + (1 + [\![A]\!]) \cdot [\![B]\!] \\
= 1 + [\![A]\!] + [\![B]\!] + [\![B]\!] + [\![A]\!] \cdot [\![B]\!] \\
= 1 + [\![A]\!] + [\![A]\!] \cdot [\![B]\!]
\end{gathered}$$

Let p, q be the arithmetic valuations of P, Q respectively. Note the following two equivalencies hold:

$$\begin{gathered}
P \vDash Q \Leftrightarrow P \rightarrow Q \text{ is a tautology} \Leftrightarrow p \leq q \\
P \equiv Q \Leftrightarrow p = q
\end{gathered}$$

where $\leq$ is a binary relation between 0 and 1 which can be fully defined in three cases:

$$\begin{gathered}0 \leq 0 \\ 0 \leq 1 \\ 1 \leq 1\end{gathered}$$

As an exercise, consider how to state the expressive completeness of $\{\neg, \wedge\}$ in this representation.
Note: from now one, we may write $\bot$ as 0 and $\top$ as 1 (mainly for convenience).