6 Logical Metatheory for Propositional Modal Logic

6.1 Metatheory Generally

Generally, a logical system or logical calculus (sometimes just a "logic") is fully described by giving three things:

1. The specification of a "syntax", i.e., a description of the signs that make up the language in question and the rules governing which ways of combining them yield well-formed sentences and other expressions, and which do not.

2. The specification of a "semantics", or theory describing the meaning or truth conditions of the sentences of the language, usually couched in terms of models, and thereby, definitions of such logical notions as validity, satisfiability, and so on.

3. The characterization of a deductive system capturing what is taken to be correct reasoning making use of the logical system in question.

Generally speaking these things cannot be described in the language of the logical system itself, but must be discussed within another language.

The object language is the language of the logical system being investigated.

The object language for all our studies this semester so far has been the language of propositional modal logic, i.e., the language of propositional logic supplemented with the operators '$\Box$' and '$\Diamond$'.

The metalanguage is the language used to describe, discuss explain and prove things about the object langauge.

In our case, the metalanguage is English, or perhaps, English supplemented with some technical jargon and symbolism ("model", "operator", "$\vDash$", etc.)

We have already described in some detail how to carry about deductions or proofs *within* various systems for modal logic: K, T, B, S4, S5, etc.

For various reasons, however, it may be worthwhile to prove things *about* logical systems. These proofs are carried out not in a given object language, but within our metalanguage. They could be called "metaproofs".

Taken together, the examination of what can be proven *about* a given logical system is called its *metatheory.*

We shall focus on our metatheory on the so-called "soundness" and "completeness" results: i.e., that whenever a conclusion can be deduced from certain premises in our deductive system, the corresponding argument is semantically valid, and vice-versa.

You might wonder what exactly the proper structure and inference rules are for our metaproofs. In general, we shall not attempt to be systematic here. While we shall sometimes identify and number "steps" in a metaproof, and be clear what follows from what, we shall not formalize these rules in any detail.

(Doing so with substantial rigor would begin an obvious infinite regress.)

In general, we shall simply allow ourselves the proof techniques taken for granted in contemporary mathematics. This includes metalanguage analogues of all the standard rules for quantified and propositional logic, as well as some more sophisticated techniques such as mathematical induction and some rudiments of set theory.

Thus, for example, corresponding to the object language rule of (MP), we will allow ourselves to reason like this.

If yadda yadda, then blah blah blah.
Yadda yadda.
____
Therefore, blah blah blah.

In the metalanguage, however, we write "if ... then ..." rather than "$\to$" to avoid any possible confusion between meta- and object language.

Our present focus is simply to establish the following relationships (for various notions of validity and deductive systems).

$$\begin{tikzcd}
A_1, A_2, \ldots, A_n \vdash B \arrow[rr, Rightarrow, "\text{(soundness)}"'] & & A_1, A_2, \ldots, A_n \vDash B \arrow[dl, Rightarrow, "\text{(tree modeling)}"] \\
& \text{The tree for } A_1, A_2, \ldots, A_n, {\sim}B \text{ closes} \arrow[ul, Rightarrow, "\text{(tree conversion)}"] &
\end{tikzcd}$$

(The two relationships at the bottom, put together, yield completeness.)

We may not be able to give the proofs of all these in rigorous detail; it may suffice if we simply understand why they hold more informally.

We begin with the relationship on the left side of this triangle.

6.2 Converting Trees to Proofs

Our first task is to show that if all branches for a given tree for checking the K-validity of an argument $A_1, A_2, \ldots, A_n$ /∴ B close, then a proof can be constructed in system K from $A_1, A_2, \ldots, A_n$ to B. We do this by showing that the tree can be *converted* into a proof by a mechanical procedure.

Given the great similarity between tree rules and inference rules, the point may seem somewhat obvious, but it is worth going over the details.

A tree for the argument is headed by the assumptions $A_1, A_2, \ldots, A_n$ and $\sim B$ in a given world (typically "**w**"), and if closed, ends in '$\bot$'. The corresponding proof takes the form of an (IP) with $A_1, A_2, \ldots, A_n$ as initial premises or hypotheses and $\sim B$ an additional assumption for *reductio*.

The goal is to convert the remainder of the tree into a proof of '$\bot$' within this subderivation, yielding an indirect proof of B:

```
w | A1
  | A2
  | ⋮
  | An
  |----
  | | ~B
  | |----
  | | ⋮
  | | ⊥
  | B          (IP)
```

The non-branching rules for propositional operators correspond rather obviously to inference rules:

- The tree rule for changing $\sim\sim A$ to A corresponds to (DN).
- The tree rule for going from A and $\sim A$ to $\bot$ corresponds to ($\bot$In).
- The tree rule for $\sim(A \rightarrow B)$ corresponds to ($\rightarrow$F).
- The tree rule for $A \,\&\, B$ corresponds to (&Out).
- The tree rule for $\sim(A \vee B)$ corresponds to a combination of (DM) and (&Out).

More interesting is the conversion process for branching rules. The first thing to note is that each statement form that leads to a branch is equivalent to, and yields in system PL, a disjunction:

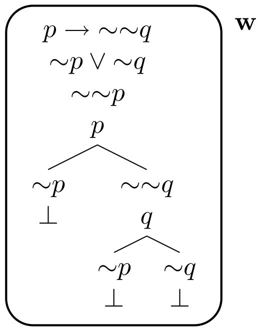
- $A \vee B$ is already in the proper form.
- $A \rightarrow B$ yields $\sim A \vee B$ (by (DN) and (Def $\vee$)).
- $\sim(A \,\&\, B)$ yields $\sim A \vee \sim B$ (by (DM)).
- $A \leftrightarrow B$ yields $(A \,\&\, B) \vee (\sim A \,\&\, \sim B)$ and $\sim(A \leftrightarrow B)$ yields $(A \,\&\, \sim B) \vee (\sim A \,\&\, B)$.

This transition made, the two branches underneath can be seen as the two subproofs making up a proof by cases (or as Garson calls, it ($\vee$Out)).

Since all branches of a closed tree must close, the goal then becomes to convert each branch into a proof of '$\bot$' for each case

Branches within branches require proofs by cases within proofs by cases.

Here's an example from an earlier handout:



Skipping (Reit) steps, this converts to:

```
 1.  w | p → ~~q
 2.    | ~p ∨ ~q
       |----
 3.    | | ~~p
       | |----
 4.    | | p              3 (DN)
 5.    | | ~p ∨ ~~q       1 (Impl)
 6.    | | | ~p
       | | |----
 7.    | | | ⊥            4,6 (⊥In)
 8.    | | | ~~q
       | | |----
 9.    | | | q            8 (DN)
10.    | | | | ~p
       | | | |----
11.    | | | | ⊥          4,10 (⊥In)
12.    | | | | ~q
       | | | |----
13.    | | | | ⊥          9,12 (⊥In)
14.    | | | ⊥            2,10–11,12–13 (∨Out)
15.    | | ⊥              5,6–7,8–14 (∨Out)
16.    | ~p               3–15 (IP)
```

This isn't even *close* to the shortest or most eloquent proof, which would be something more like:

```
1.  │ p → ∼∼q
2.  │ ∼p ∨ ∼q
    ├──
3.  │ │ p
    │ ├──
4.  │ │ ∼∼q        1,3 (MP)
5.  │ │ ∼p         2,4 (DS)
6.  │ │ ⊥          3,5 (⊥In)
7.  │ ∼p           3–7 (IP)
```

However, the procedure generating the longer proof from the tree was entirely mechanical. Indeed, constructing a tree and then converting it provides a recipe for generating a proof if there is one.

To convert any K-trees, we need only add methods for converting the tree rules for statements that begin with '$\Box$' or '$\Diamond$' and their negations.

For their negations, it is simply worth noting that a statement of the form $\sim\Diamond A$ is treated precisely like $\Box\sim A$ with regard to trees, and one of the form is $\sim\Box A$ is treated precisely like $\Diamond\sim A$. Since we have the derived rules ($\sim\Box$) and ($\sim\Diamond$) for swapping them in proofs, we shall do so, and hence focus all our attention on the rules for unnegated $\Box$- and $\Diamond$-statements.

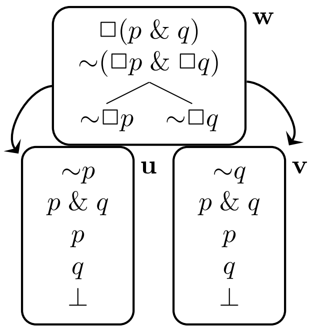
The tree rule for $\Diamond A$ creates a new world accessible from the given world in which A is true. In converting, this corresponds to a new *world-subproof* (see page 7 of the handouts) headed by $\Box, A$.

The tree rule for statements of the form $\Box A$ allows us to introduce A into an already present world box accessible from the world where we found $\Box A$. The proof conversion of this is simply an application of the ($\Box$Out) rule to bring A into a world-subproof where we had $\Box A$ prior to that subproof.

Strictly speaking, if "$\bot$" is derived in a world-subproof rather than in our original subproof headed by the reductio assumption, this only gives us "$\Diamond\bot$" outside of that world subproof by means of ($\Diamond$Out). However, this eventually yields "$\bot$" in the main subproof owing to the fact that $\vdash_K \sim\Diamond\bot$. We shall abbreviate this as follows:

```
◇A
│ □, A
├──
│ ⋮
│ ⊥
⊥            (◇⊥)
```

This gives us all we need to convert this tree from an earlier handout:



```
1.   w│ □(p & q)
      ├──
2.    │ │ ∼(□p & □q)
      │ ├──
3.    │ │ ∼□p ∨ ∼□q          2 (DM)
4.    │ │ │ ∼□p
      │ │ ├──
5.    │ │ │ ◇∼p              4 (∼□)
6.    │ │ │ u│ □, ∼p
      │ │ │  ├──
7.    │ │ │  │ p & q         1 (□Out)
8.    │ │ │  │ p             7 (&Out)
9.    │ │ │  │ q             7 (&Out)
10.   │ │ │  │ ⊥             6,8 (⊥In)
11.   │ │ │ ⊥                5, 6–10 (◇⊥)
12.   │ │ │ ∼□q
      │ │ ├──
13.   │ │ │ ◇∼q              12 (∼□)
14.   │ │ │ v│ □, ∼q
      │ │ │  ├──
15.   │ │ │  │ p & q         1 (□Out)
16.   │ │ │  │ p             15 (&Out)
17.   │ │ │  │ q             15 (&Out)
18.   │ │ │  │ ⊥             14,17 (⊥In)
19.   │ │ │ ⊥                13, 14–18 (◇⊥)
20.   │ │ ⊥                  3,4–11,12–19 (∨Out)
21.   │ □p & □q              2–20 (IP)
```

You may notice that here I have tagged the world subproofs with the corresponding world label ("**u**" and "**v**")from the tree (and the main subproof with "**w**"). This is simply to make the conversion easier to follow. Untagged subproofs are not world subproofs (or boxed subproofs of any sort), and hence the (Reit) rule always traverses past them.