❖ *Proofs and Deductions* ❖

3.35. Fundamentals of Deduction

> "'You reasoned it out so beautifully' I exclaimed in unfeigned admiration. 'It is so long a chain, and yet every link rings true.'"
>
> – Doctor Watson, in Arthur Conan Doyle, **The Adventures of Sherlock Holmes**

Though they mark a significant departure from the earlier semantic methods, **deductions** provide another means for formally demonstrating argument validity – as well for achieving other results familiar from our earlier semantic approach. And like the semantic procedures, deductions can even trace their roots back to certain features of informal logic.

In general, formal logic aims to develop a test of validity which agrees with our intuitive grasp of simple, obvious arguments, but 'scales up' to manage arbitrarily large arguments outstripping those intuitions. With deductions, an inspiration for handling complex cases on the basis of simple ones comes from the **construction rules** of the formal language.

Because they are **recursive** – capable of 'recycling' – those rules generate an infinite number of formal sentences using only finite resources. The trick was to *repeatedly* apply a limited number of procedures to a (growing) set of accepted items. From an initial stock, those procedures generate new items which are then recycled as fresh input – and so on.

In the construction rules, the items accepted at the outset are the sentence letters – the *most basic* sort of formal sentence. The procedures applied to those items are the three molecular rules – the Negation, Conjunction, and Disjunction Rules. They are *recursive* rules because the output of any of them can be 'recycled' as new input for any of them.

So in staking out the entire family of formal sentences, construction began by accepting the sentence letters as formal sentences.

> Set of accepted sentences: {P, Q, R, …}

The Negation Rule would then draw a member of this set – say, "P" – as input, yielding "~P" as a new grammatical sentence.

Set of accepted sentences: {P, Q, R, **~P**, …}

Since the recursive rules can take *any* formal sentence as input, the Negation Rule is then free to draw "~P" from this set as a new input, yielding the further sentence "~~P".

Set of accepted sentences: {P, Q, R, ~P, **~~P**, …}

And so on.

The same strategy can be applied to generate an infinite number of logically acceptable arguments – all the **valid arguments** in that logical language. With any such valid argument, we can begin with a set of **accepted** sentences – the premises – and apply a finite stock of procedures to this set to generate further accepted sentences.

The '**procedures'** in this case will be uncontroversially valid argument forms – where the premises of that argument form act as the 'input', and its conclusion serves as 'output'.

The following is an uncontroversial example of a valid argument pattern.

(● ∨ ▲)
~●
———————
∴ ▲

Since it begins with a disjunction, and extracts from it one of the parts, we call this argument pattern **Vel Elimination** – or "**Vel Elim**" ("∨**-Elim**") for short.

We recognize any instance of ∨-Elim as a valid argument – such as the following familiar English example.

1. Either the Chess Club won the prize, or the Surf Club won the prize.
2. The Chess Club did not win the prize.

(So,) The Surf Club won the prize.

But when used recursively, this same argument pattern allows us to recognize the validity of more complicated arguments as well. For instance, the following English argument is a bit more complex.

1. We’ll have either ice cream or cake, or we’ll have pie.
2. We won’t have pie.
3. We won’t have cake.

∴ We’ll have ice cream.

Perhaps it’s still simple enough that your intuitions can judge its validity immediately. But suppose there were someone whose intuitions were very quickly boggled by complexity: he can recognize ∨-Elim as a valid pattern, but is stumped by this longer argument. Using *only* ∨-Elim, in a recursive fashion, you can show such a logically myopic friend that this larger argument is indeed valid.

Validity being an issue of logical form, we first use a translation table to translate the argument into formal language.

P: We’ll have ice cream
Q: We’ll have cake
R: We’ll have pie

1. ((P ∨ Q) ∨ R)
2. ~R
3. ~Q

∴ P

The premises are the only sentences accepted so far.

Set of accepted sentences: { ((P ∨ Q) ∨ R), ~R, ~Q }

But with the first two premises we have a disjunction "((P ∨ Q) ∨ R)," and the negation of its right part, "~R".

1. **((P ∨ Q) ∨ <u>R</u>)** ⇐ Disjunction
2. **<u>~R</u>** ⇐ Negation of Right Part

Recognizing this as an instance of ∨-Elim, we see as well that the left part of the disjunction, "(P ∨ Q)," follows validily.

((P ∨ Q) ∨ <u>R</u>) ⇐ Disjunction
<u>~R</u> ⇐ Negation of Right Part

(P ∨ Q) ⇐ Left Part

So "(P ∨ Q)" is added to our set of accepted sentences

Set of accepted sentences: { ((P ∨ Q) ∨ R), ~R, ~Q, **(P ∨ Q)** }

The last two sentences in this set are the disjunction "(P ∨ Q)" and negation of its right part, "~Q". This is just another instance of ∨-Elim: since these two sentences are accepted as true, ∨-Elim directs us to accept as well the left half of that disjunction – "P".

Set of accepted sentences: { ((P ∨ Q) ∨ R), ~R, **~Q**, **(P ∨ Q)** }

(<u>P</u> ∨ Q) ⇐ Disjunction
~Q ⇐ Negation of Right Part

<u>P</u> ⇐ Left Part

Hence "P" is added to the set of accepted sentences as well.

Set of accepted sentences: { ((P ∨ Q) ∨ R), ~R, ~Q, (P ∨ Q), **P** }

We’ve demonstrated that if the first three sentences are accepted as true, “P” should be as well. So the original logical form is indeed **valid**.

That means: anyone accepting ∨-Elim as valid should also recognize this argument form as valid.

1. ((P ∨ Q) ∨ R)
2. ~R
3. ~Q

———————

∴ P

Starting from a set of accepted premises, we collected a series of new sentences using only accepted argument patterns (in this case just one) to demonstrate the validity of a logical form. We call such a procedure a **deduction**: we **deduced** the conclusion “P” from the original three premises.

And just as sentence construction needed only a few recursive rules to build an infinite number of formal sentences, we can likewise demonstrate the validity of any valid argument in the logical language, using a finite stock of valid argument patterns – ∨-Elim, and other uncontroversial logical forms – in a **recursive** fashion.[1]

	Construction	Deduction
Original Set of Accepted Sentences:	Sentence Letters	Premises of the Argument
Rules:	Construction Rules (Rules 2, 3, and 4)	∨-elim (and Other Uncontroversial Logical Forms)
Final Output:	Formal Sentence being constructed	Conclusion of the Argument

[1] In some logic texts the construction rules are called “formation rules,” while those for deduction are called “transformation rules”.

We said at the outset that deductions find their origins in informal logic. Now we see how: deductions are really just a formal extension of **chain arguments**.

Just as a chain argument reaches a sub-conclusion only to use it as a premise supporting a further conclusion, so deductions accept the conclusion of an argument form only to then use it as further input for an argument form. And just as a chain argument was only as valid as its weakest link – so that with a chain of valid arguments, the entire chain inherits that validity – so deductions show that an argument form is valid by tracing a *chain* of valid links from premises to conclusion.

We close with a bit of bookkeeping.

It will prove convenient to trade in the "set of accepted sentences" for a simple vertical list, with an account of why each sentence in the list is accepted. The above deduction then begins with just the premises listed.

1. $((P \vee Q) \vee R)$
2. $\sim R$
3. $\sim Q$

} Premises

To keep in mind the conclusion we're seeking to deduce – here, "P" – we add a memo to *get* that sentence. (This isn't a line of the deduction like the premises are – just a reminder on the side; so we don't number this "Get" line.)

1. $((P \vee Q) \vee R)$
2. $\sim R$
3. $\sim Q$

} Premises

———————— Get: P

Each inference from there is listed with its justification on the right: the argument pattern used, and the lines that were its input (premises).

1. ((P ∨ Q) ∨ R) } Premises
2. ~R
3. ~Q

——————————Get: P

4. (P ∨ Q) 1, 2, ∨-elim
5. P 3, 4, ∨-elim

Once we’ve deduced the conclusion, we cross out the “Get” line – like checking it off our list of things to do. The deduction is then complete.

1. ((P ∨ Q) ∨ R) } Premises
2. ~R
3. ~Q

——————————~~Get~~: P

4. (P ∨ Q) 1, 2, ∨-elim
5. P 3, 4, ∨-elim