Chapter 5:
Working with Proofs

In the last chapter we introduced the syntactic notion of a formal deductive proof in Propositional Logic and wrote code that checked whether a given alleged proof is indeed a syntactically valid proof. In this chapter we proceed to mathematically analyze such syntactic formal proofs, and prove theorems about formal proofs. Notice the two very different uses of "proof" in the last sentence: in "proving theorems. . . ," the meaning is the usual mathematical notion of presenting a mathematically convincing, but not syntactically formal, argument of correctness; in ". . . about formal proofs" we mean syntactically formal propositional proofs. The former is usually done in mathematical language but in this book will be achieved by writing Python programs with the claimed functionality, while the latter are usually viewed as mathematical objects (formally, strings of symbols), but we will use Python objects of class `Proof`, handled by said programs, to represent them. Disambiguating these two very different notions of "proof" by defamiliarizing them from what we intuitively grasp as a proof, turning one of them into "programming" and the other into "object of class `Proof`," is at the heart of the pedagogical approach of this book, and we hope that it will help avoid the confusion that is sometimes stirred by this ambiguity and by our intuition of what a proof is.

While one of our long-term goals in this book is to be able to completely formalize mathematically convincing proofs of the former kind as syntactically formal proofs of the latter kind, and thus to explain why there is in fact no ambiguity here and that both are called "proof" for a good reason, this will have to wait for the stronger **predicate logic** studied in the second half of this book. For now, we are happy with using "regular" Mathematics (or programming) to prove claims regarding our formal syntactic propositional proofs (or equivalently, the corresponding `Proof` objects).

1 Using Lemmas

Recall that each of our formal proofs has a well-defined set of inference rules that it can use, and each step (line) in the proof has to apply one of these rules. In normal Mathematics we often abstract a sub-argument as a **lemma**, prove the lemma once, and from that point allow ourselves to continue using the lemma as an additional inference rule without needing to reprove it every time that we use it in some other proof. In fact, the way that Mathematics progresses is exactly by proving a richer and richer set of lemmas and theorems, all of them to be used "as inference rules" later on at will. This seems to be such a basic element of mathematical reasoning, that one may well expect such a capability of defining lemmas to be an integral part of our definition of formal proofs (just as high-level programming languages have the ability to define and use functions and methods defined as part of the language). As we will now see, we do not need to define this capability, and in fact having the capability of defining lemmas

is only for the *human* convenience of writing proofs, while our formal proofs can mimic this ability without having it be part of the definition. (To some extent, this is like machine languages often do not have any notion of a procedure or function call, and yet can implement that functionality when higher-level languages are compiled into such machine languages.)

The first thing that we have to verify is that if some lemma is proven, then specializations of that lemma can also be proven. In normal Mathematics we take this as a triviality of course (every first-year Mathematics or Computer Science student has at some point applied a lemma that is phrased for any function f to a specific function g, treating both the assumptions and the conclusion of the lemma as if they were originally phrased for this specific g rather than for f), but once we have limited ourselves to formal proofs of the very specific form defined in the previous chapter, we need to show that this property holds for this specific form of formal proofs.

Task 1. Implement the missing code for the function `prove_specialization(proof, specialization)` (in the file `propositions/proofs.py`), which takes a (valid) deductive proof of some inference rule, and returns a deductive proof of the given specialization of that rule, via the same inference rules used in the original proof.

propositions/proofs.py

```
def prove_specialization(proof: Proof, specialization: InferenceRule) -> Proof:
    """Converts the given proof of an inference rule into a proof of the given
    specialization of that inference rule.

    Parameters:
        proof: valid proof to convert.
        specialization: specialization of the conclusion of the given proof.

    Returns:
        A valid proof of the given specialization via the same inference rules
        as the given proof.
    """
    assert proof.is_valid()
    assert specialization.is_specialization_of(proof.statement)
    # Task 5.1
```

Example: If `proof` is a proof of the inference rule with assumptions '(p|q)', '(~p|r)' and conclusion '(q|r)', then `specialization` could be, e.g., the inference rule with assumptions '(x|p)', '(~x|(y&z))' and conclusion '(p|(y&z))'.
Hint: Start by using the `specialization_map()` method of class `InferenceRule` to get the map that specializes the rule that `proof` proves into `specialization`.

Your solution to Task 1 proves the following lemma:

Lemma (Specialization Provability)**.** *Let $\mathcal{R}$ be a set of inference rules. Every specialization of every lemma that can be formally proven via $\mathcal{R}$ can itself be formally proven via $\mathcal{R}$ as well.*

With the above lemma in hand, we are now ready to show that the formal proof system that we defined in the previous chapter does indeed have the power to allow for gradual building of lemmas and their usage—a result that we will formally state below and call the **Lemma Theorem**.

Task 2 (Programmatic Proof of the Lemma Theorem)**.** In this task you will implement the missing code for the function `inline_proof(main_proof, lemma_proof)`, which takes a deductive proof `main_proof` of some inference rule via a set of inference rules $\mathcal{R}$, takes a deductive proof `lemma_proof` of one of the inference rules $\lambda \in \mathcal{R}$ (short for lemma) via a set of inference rules $\mathcal{R}'$ (where $\lambda \notin \mathcal{R}'$ of course), and returns a deductive proof of the inference rule proved by `main_proof`, however via the inference rules $(\mathcal{R} \setminus \{\lambda\}) \cup \mathcal{R}'$.

a. First implement the missing code for the function `inline_proof_once(proof, line_number, lemma_proof)`, which replaces only a single instance of the use of the given lemma, and thus the returned proof is allowed to use all of the inference rules in $\mathcal{R} \cup \mathcal{R}'$, including λ, but must use λ one time less than in the given proof.

propositions/proofs.py

```
def inline_proof_once(main_proof: Proof, line_number: int, lemma_proof: Proof) \
    -> Proof:
    """Inlines the given proof of a "lemma" inference rule into the given proof
    that uses that "lemma" rule, eliminating the usage of (a specialization of)
    that "lemma" rule in the specified line in the latter proof.

    Parameters:
        main_proof: valid proof to inline into.
        line: index of the line in `main_proof` that should be replaced.
        lemma_proof: valid proof of the inference rule of the specified line (an
            allowed inference rule of `main_proof`).

    Returns:
        A valid proof obtained by replacing the specified line in `main_proof`
        with a full (specialized) list of lines proving the formula of the
        specified line from the lines specified as the assumptions of that line,
        and updating line indices specified throughout the proof to maintain the
        validity of the proof. The set of allowed inference rules in the
        returned proof is the union of the rules allowed in the two given
        proofs, but the "lemma" rule that is used in the specified line in
        `main_proof` is no longer used in the corresponding lines in the
        returned proof (and thus, this "lemma" rule is used one less time in the
        returned proof than in `main_proof`).
    """
    assert main_proof.lines[line_number].rule == lemma_proof.statement
    assert lemma_proof.is_valid()
    # Task 5.2a
```

Guidelines: Implement the easiest implementation, which does not generate a proof with a minimal number of lines but does avoid a lot of clutter, by returning a proof that contains three batches of lines:

1. All the lines of `proof` up to line `line_number-1`, unmodified.
2. All the lines of a proof of the specialization (of the lemma) that is used in line `line_number` of `proof`, only with the following two modifications:
   (a) All indices should be shifted by the same proper number.
   (b) Each line that corresponds to any assumption of the specialization that is not an assumption of `proof` should be modified by adding an appropriate rule and an appropriate assumption list (these can simply be copied from the line that is used as the corresponding assumption in line `line_number` of `proof`).

3. All the lines of `proof` from line `line_number+1`, only with all indices that originally point to line `line_number` or greater shifted by the same proper number.

b. Now you can implement the function `inline_proof(main_proof, lemma_proof)` by using the above repeatedly, until all uses of the given lemma in the original proof are eliminated.

propositions/proofs.py

```
def inline_proof(main_proof: Proof, lemma_proof: Proof) -> Proof:
    """Inlines the given proof of a "lemma" inference rule into the given proof
    that uses that "lemma" rule, eliminating all usages of (any specialization
    of) that "lemma" rule in the latter proof.

    Parameters:
        main_proof: valid proof to inline into.
        lemma_proof: valid proof of one of the allowed inference rules of
            `main_proof`.

    Returns:
        A valid proof obtained from `main_proof` by inlining (an appropriate
        specialization of) `lemma_proof` in lieu of each line that specifies the
        "lemma" inference rule proved by `lemma_proof` as its justification. The
        set of allowed inference rules in the returned proof is the union of the
        rules allowed in the two given proofs but without the "lemma" rule
        proved by `lemma_proof`.
    """
    # Task 5.2b
```

Your solution to Task 2 proves the following theorem, which formalizes the fact that explicitly allowing the usage of lemmas in our definition of a deductive proof would not have made deductive proofs any more powerful.

Theorem (The Lemma Theorem)**.** *If there exists a formal proof of a lemma A via a set of inference rules $\mathcal{R} \cup \{\lambda\}$ and there exists a formal proof of λ via $\mathcal{R}$, then there exists a formal proof of A via only the set $\mathcal{R}$.*

2 Modus Ponens

Having developed our general understanding of proofs as far as we could without assuming anything about the inference rules that our proofs are allowed to use, it is time to start introducing the inference rules that will form our set of standard allowed inference rules — our **axiomatic system**. Arguably the most natural and basic sound inference rule, which was already known at the 3rd century BCE, is the **Modus Ponens**, or **MP** for short, inference rule, to which you were already briefly introduced in the final task of the previous chapter.

Modus Ponens (MP): Assumptions: 'p', '(p→q)'; Conclusion: 'q'

propositions/axiomatic_systems.py

```
# Axiomatic inference rules that only contain implies

#: Modus ponens / implication elimination
MP = InferenceRule([Formula.parse('p'), Formula.parse('(p->q)')],
```

```
                Formula.parse('q'))
                       ⋮
```

It is straightforward to verify that MP is indeed sound, and your implementation of `is_sound_inference()` has in fact already done so: one of the ways in which we have tested it is by making sure that it successfully verifies the soundness of MP.

In the final task of the previous chapter, MP allowed you to logically proceed from having proven ϕ (called the **antecedent** of MP) and from '$(\phi \to \psi)$' (called the **conditional claim** of MP) being (a specialization of) an allowed inference rule, to having proven ψ (called the **consequent** of MP). In fact, you have sequentially applied MP *twice* to logically proceed from having proven ϕ and ψ and from '$(\phi \to (\psi \to \xi))$' being an allowed inference rule, to having proven ξ. Even more generally, MP similarly allows us to progress from having proven each of $\phi_1, \ldots, \phi_n$ and from '$(\phi_1 \to (\phi_2 \to \cdots (\phi_n \to \xi) \cdots))$' being an allowed inference rule to having proven ξ. As the former two proof techniques (that correspond to the cases $n = 1$ and $n = 2$) are extremely handy and common, your first task is to implement them in a generic way as two functions that will save you the trouble of repeating these two techniques over and over again in many of the proofs that you will construct in this and in the next chapter.

Task 3.

a. Start by implementing the missing code for the function `prove_corollary(antecedent_proof, consequent, conditional)`, which takes a proof of some "antecedent" formula ϕ (from some list of assumptions and using some set of inference rules), a desired `consequent` ψ, and an assumptionless inference rule `conditional` of which '$(\phi \to \psi)$' is a specialization, and returns a proof of ψ from the assumptions of the original proof, via the inference rules of the original proof, MP, and `conditional`.

propositions/deduction.py

```
def prove_corollary(antecedent_proof: Proof, consequent: Formula,
                    conditional: InferenceRule) -> Proof:
    """Converts the given proof of a formula `antecedent` into a proof of the
    given formula `consequent` by using the given assumptionless inference rule
    of which '(`antecedent`->`consequent`)' is a specialization.

    Parameters:
        antecedent_proof: valid proof of `antecedent`.
        consequent: formula to prove.
        conditional: assumptionless inference rule of which the assumptionless
            inference rule with conclusion '(`antecedent`->`consequent`)' is a
            specialization.

    Returns:
        A valid proof of `consequent` from the same assumptions as the given
        proof, via the same inference rules as the given proof and in addition
        `MP` and `conditional`.
    """
    assert antecedent_proof.is_valid()
    assert InferenceRule([],
                         Formula('->', antecedent_proof.statement.conclusion,
                                 consequent)).is_specialization_of(conditional)
    # Task 5.3a
```

b. Now, implement the missing code for the slightly more complex variant function `combine_proofs(antecedent1_proof, antecedent2_proof, consequent, double_conditional)`, which takes *two* proofs of two "antecedent" formulas ϕ_1 and ϕ_2, both from the same[1] list of assumptions and using the same set of inference rules, a desired `consequent` ψ, and an assumptionless inference rule `double_conditional` of which '$(\phi_1 \to (\phi_2 \to \psi))$' is a specialization, and, as before, returns a proof of ψ from the assumptions of the two original proofs, via the inference rules of the two original proofs, MP, and `conditional`.

propositions/deduction.py

```
def combine_proofs(antecedent1_proof: Proof, antecedent2_proof: Proof,
                   consequent: Formula, double_conditional: InferenceRule) -> \
        Proof:
    """Combines the given proofs of two formulas `antecedent1` and `antecedent2`
    into a proof of the given formula `consequent` by using the given
    assumptionless inference rule of which
    '(`antecedent1`->`antecedent2`->`consequent`)' is a specialization.

    Parameters:
        antecedent1_proof: valid proof of `antecedent1`.
        antecedent2_proof: valid proof of `antecedent2` from the same
            assumptions and inference rules as `antecedent1_proof`.
        consequent: formula to prove.
        double_conditional: assumptionless inference rule of which the
            assumptionless inference rule with conclusion
            '(`antecedent1`->`antecedent2`->`consequent`)' is a specialization.

    Returns:
        A valid proof of `consequent` from the same assumptions as the given
        proofs, via the same inference rules as the given proofs and in addition
        `MP` and `conditional`.
    """
    assert antecedent1_proof.is_valid()
    assert antecedent2_proof.is_valid()
    assert antecedent1_proof.statement.assumptions == \
           antecedent2_proof.statement.assumptions
    assert antecedent1_proof.rules == antecedent2_proof.rules
    assert InferenceRule(
        [], Formula('->', antecedent1_proof.statement.conclusion,
        Formula('->', antecedent2_proof.statement.conclusion, consequent))
        ).is_specialization_of(double_conditional)
    # Task 5.3b
```

3 The Deduction Theorem

Our next order of business on our "working with proofs" agenda is understanding the connection between the two notions of $\{\phi\} \vdash \psi$ and $\vdash$ '$(\phi \to \psi)$' (where we have the same set of inference rules $\mathcal{R}$ in both expressions, so $\vdash$ is really shorthand for $\vdash_{\mathcal{R}}$ in both of these expressions). First let us recall what each of these two notions formally means. $\{\phi\} \vdash \psi$ means that there is a formal proof of the formula ψ from the single assumption ϕ (via

[1] We require that the assumptions of the two given proofs are exactly the same and in the same order for simplicity and since this is what we will use, even though the same solution would also work with no requirements on the assumptions, and only returning a proof from all the assumptions that are used in any of the given proofs.

the inference rules $\mathcal{R}$). On the other hand, $\vdash$ '$(\phi\to\psi)$' means that there is a formal proof of the single formula '$(\phi\to\psi)$' from no assumptions (again, via the inference rules $\mathcal{R}$).

While this question of the connection between these two notions is purely syntactic, a good intuition for why there should be some connection comes from the analogous semantic question: what is the relation between $\{\phi\} \models \psi$ and $\models$ '$(\phi\to\psi)$'? Here the answer is that these two are equivalent: $\{\phi\} \models \psi$ (i.e., ϕ entails ψ, or equivalently, the rule with single assumption ϕ and conclusion ψ is sound) means that in every model in which ϕ is satisfied, also ψ is satisfied. But this exactly means that the formula '$(\phi\to\psi)$' is a tautology, i.e., that $\models$ '$(\phi\to\psi)$'.

So we now go back to the syntactic question of $\{\phi\} \vdash \psi$ and $\vdash$ '$(\phi\to\psi)$'. In this section you will show that, under some conditions on the set of inference rules $\mathcal{R}$, indeed we can make sure that these two notions coincide, a fact that will be quite useful also in the next chapter. We will start by exploring the easier direction of when $\vdash$ '$(\phi\to\psi)$' implies $\{\phi\} \vdash \psi$, after which we will turn to explore the harder converse direction, of when $\{\phi\} \vdash \psi$ implies $\vdash$ '$(\phi\to\psi)$'.

The first condition that we will impose on our set of inference rules $\mathcal{R}$ is that it contains MP. Using MP, we can immediately see the easy side of the above equivalence: that $\vdash$ '$(\phi\to\psi)$' implies $\{\phi\} \vdash \psi$. Indeed, one way to prove the conclusion ψ from the assumption ϕ is, analogously to your implementation of `prove_corollary`, to take the proof for '$(\phi\to\psi)$', append an assumption line containing the assumption ϕ, and directly apply MP to the conclusions of the last two lines (i.e., to ϕ and to '$(\phi\to\psi)$') to deduce ψ. This simple technique continues to holds even regardless of any other assumptions that the original proof of '$(\phi\to\psi)$' uses. (These assumptions are simply used by the new proof as well.) This proves the following lemma, which states the "easy direction" of the equivalence that we are after:

Lemma. *Let $\mathcal{R}$ be a set of inference rules that includes MP, let A be an arbitrary set of formulas, and let ϕ and ψ be two additional formulas. If $A \vdash_{\mathcal{R}}$ '$(\phi\to\psi)$', then $A \cup \{\phi\} \vdash_{\mathcal{R}} \psi$.*

We now move to the hard part of the equivalence: showing that $\{\phi\} \vdash \psi$ implies $\vdash$ '$(\phi\to\psi)$', and even more generally, that $A \cup \{\phi\} \vdash \psi$ implies $A \vdash$ '$(\phi\to\psi)$' for any set of additional assumptions A. This result is called the **Deduction Theorem**, and its proof is a significant milestone toward the goal of the first part of this book. We will prove this theorem for any set $\mathcal{R}$ of inference rules that also contains, in addition to Modus Ponens, the following three **axioms** (we will use the term **axioms** to refer to the assumptionless inference rule that will be part of our axiomatic system — our set of allowed inference rules), to which you were also already introduced in the final task of the previous chapter:

I0: '(p→p)'

I1: '(q→(p→q))'

D: '((p→(q→r))→((p→q)→(p→r)))'

propositions/axiomatic_systems.py

```
# Axiomatic inference rules that only contain implies
                          ⋮
#: Self implication
I0 = InferenceRule([], Formula.parse('(p->p)'))
#: Implication introduction (right)
```

```
I1 = InferenceRule([], Formula.parse('(q->(p->q))'))
#: Self-distribution of implication
D = InferenceRule([], Formula.parse('((p->(q->r))->((p->q)->(p->r)))'))
```

It is straightforward to verify that these three axioms (and the additional axioms that will be presented in the remainder of this chapter and in the next chapter) are sound, and once again, your implementation of `is_sound_inference()` has in fact already done this: one of the ways in which we have tested it is by making sure that it successfully verifies the soundness of all of these axioms.

Recall that in the final task of the previous chapter you have already proved I0 via $\{\text{MP}, \text{I1}, \text{D}\}$, so by the Lemma Theorem, anything that can be proven via $\{\text{MP}, \text{I0}, \text{I1}, \text{D}\}$ can in fact be proven without I0, via only $\{\text{MP}, \text{I1}, \text{D}\}$. So, we allow ourselves to also use I0 just for convenience, even though this is in fact not really needed.

We will place one additional restriction on our set $\mathcal{R}$ of inference rules: that all of these inference rules, except MP, have no assumptions. In the strictest technical sense, this will be sufficient for our purposes since from this point on, our sets of inference rules will always have this property. On a more conceptual level, one can see that this is not a significant restriction: once we have MP in our set of inference rules, we can always replace an inference rule with assumption ϕ and conclusion ψ with the assumptionless inference rule '$(\phi \to \psi)$' without losing any "proving power," since if ϕ was already proven in our proof, by your implementation of `prove_corollary()` this new assumptionless rule (in concert with MP) immediately lets us deduce ψ as the original rule did. As already briefly mentioned in the discussion above, a rule with multiple assumptions $\phi_1, \ldots, \phi_n$ and conclusion ψ can be similarly replaced by an assumptionless rule of the form '$(\phi_1 \to (\phi_2 \to \cdots (\phi_n \to \xi) \cdots))$' without losing any "proving power."

Let us see what we have to do to show that $A \cup \{\phi\} \vdash \psi$ implies $A \vdash$ '$(\phi \to \psi)$'. We are given a proof of ψ from an assumption ϕ, as well as some other assumptions A, and need to create a new proof of '$(\phi \to \psi)$' from only the assumptions in A. How can we do this? The idea is to go over the original proof line by line, and if the ith line of the original proof contains a formula ξ_i, then to add to our new proof a line containing '$(\phi \to \xi_i)$' instead. In terms of the conclusion, this is exactly what we want since the last line of the original proof is ψ and therefore the last line of the new proof will be '$(\phi \to \psi)$', as needed. The problem is that the new list of lines need not really constitute a valid proof: it's quite possible that none of these lines is an assumption or follows from previous ones using one of the allowed inference rules. The point is that, assuming that we have the required inference rules at our disposal, we will be able to add some "intermediate lines" in between each pair of line of the new proof, to turn it into a valid proof. Let us consider which kinds of lines we can have in the original proof, and see how we can deal with each of them:

- It may be that the line ξ_i is one of the assumptions that also remains an assumption in the new proof (i.e., $\xi_i \in A$). The problem is that our strategy requires us to deduce a line with '$(\phi \to \xi_i)$' while only ξ_i is an allowed assumption. Conveniently, it is possible to deduce '$(\phi \to \xi_i)$' from ξ_i via I1 along with MP (hint: take inspiration from the implementation of `prove_corollary()`).

- It may be that ξ_i is exactly the assumption ϕ that we are no longer allowed to use in the new proof. Our strategy calls for deducing a line with '$(\phi \to \xi_i)$', which in this case is '$(\phi \to \phi)$'. Conveniently, I0 allows to deduce this *without* using ϕ as an assumption.

- It may be that ξ_i is deduced by MP from two previous lines containing some formulas ξ_j and ξ_k. We need to deduce '$(\phi \to \xi_i)$', but can rely on the fact that have already deduced have '$(\phi \to \xi_j)$' and '$(\phi \to \xi_k)$' in the new proof (in the lines corresponding to the lines containing ξ_j and ξ_k in the original proof). Conveniently, a clever usage of D along with MP allows us to deduce this (hint: take inspiration from the implementation of `combine_proofs()`).
- The last option is that ξ_i is deduced by some inference rule that is not MP. As we assume that all inference rules except MP have no assumptions, ξ_i is therefore (the conclusion of) a specialization of an allowed assumptionless inference rule. So, we can deduce ξ_i in the new proof as well, and we conveniently already know how to deduce '$(\phi \to \xi_i)$' from it via I1 along with MP.

You are now asked to implement the above proof strategy.

Task 4 (Programmatic Proof of the Deduction Theorem)**.** Implement the missing code for the function `remove_assumption(proof)`, which takes as input a deductive proof of a conclusion ψ from a nonempty list of assumptions via a set of inference rules that, except for MP, have no assumptions. The function returns a deductive proof of '$(\phi \to \psi)$', where ϕ is the last assumption of the given proof, from the assumptions of the given proof except for ϕ, via the original set of inference rules as well as MP, I0, I1, and D.

propositions/deduction.py

```
def remove_assumption(proof: Proof) -> Proof:
    """Converts a proof of some `conclusion` formula, the last assumption of
    which is an assumption `assumption`, into a proof of
    '(`assumption`->`conclusion`)' from the same assumptions except
    `assumption`.

    Parameters:
        proof: valid proof to convert, with at least one assumption, via some
            set of inference rules all of which have no assumptions except
            perhaps `MP`.

    Return:
        A valid proof of '(`assumption`->`conclusion`)' from the same
        assumptions as the given proof except the last one, via the same
         inference rules as the given proof and in addition `MP`, `I0`, `I1`,
         and `D`.
    """
    assert proof.is_valid()
    assert len(proof.statement.assumptions) > 0
    for rule in proof.rules:
        assert rule == MP or len(rule.assumptions) == 0
    # Task 5.4
```

Your solution to Task 4 proves the final, hard direction, of the Deduction Theorem.

Theorem (The Deduction Theorem for Propositional Logic)**.** *Let $\mathcal{R}$ be a set of inference rules that includes MP, I1, and D, and may additionally include only inference rules with no assumptions. Let A be an arbitrary set of formulas, and let ϕ and ψ be two additional formulas. Then $A \cup \{\phi\} \vdash_{\mathcal{R}} \psi$ if and only if $A \vdash_{\mathcal{R}}$ '$(\phi \to \psi)$'.*

Recall that while we allowed using also I0 in Task 4, since you have already shown in the last task of the previous chapter that I0 can be proven from $\{\text{MP}, \text{I1}, \text{D}\}$, then by the

Lemma Theorem, the statement of the Deduction Theorem need not assume (and indeed, does not assume) that using I0 is allowed. The Deduction Theorem is an extremely useful tool for writing proofs. Many results that are quite tedious to prove directly from the above (and other) inference rules are easily proven using the Deduction Theorem. The final task of this section demonstrates such a result.

Task 5. Prove the following inference rule: Assumptions: '(p→q)', '(q→r)'; Conclusion: '(p→r)'; via the inference rules MP, I0, I1, and D. The proof should be returned by the function `prove_hypothetical_syllogism()` (in the file `propositions/some_proofs.py`), whose missing code you should implement.

propositions/some_proofs.py

```
#: Hypothetical syllogism
HS = InferenceRule([Formula.parse('(p->q)'), Formula.parse('(q->r)')],
                   Formula.parse('(p->r)'))

def prove_hypothetical_syllogism() -> Proof:
    """Proves `HS` via `MP`, `I0`, `I1`, and `D`.

    Returns:
        A valid proof of `HS` via the inference rules `MP`, `I0`, `I1`, and `D`.
    """
    # Task 5.5
```

Indeed, while it is not very intricate to prove that 'r' holds given that 'p', '(p→q)', and '(q→r)' hold, proving the hypothetical syllogism above without using the Deduction Theorem would have been more intricate and considerably less straightforward (even for this simple example)—just examine the resulting proof that your solution returns!

4 Proofs by Contradiction

A very popular proof strategy throughout Mathematics is that of proof by way of contradiction: to prove a formula ϕ, we assume that its negation '$\sim\phi$' holds, prove some **inconsistency** from it, for example that ψ holds even though we know that '$\sim\psi$' holds, and then deduce from this that our assumption '$\sim\phi$' was flawed, that is, that ϕ in fact holds. As it turns out, while this proof strategy seems at first glance to be quite far from the notion of deductive proofs that we have defined in the previous chapter, the ability to prove by way of contradiction in fact, as we will see in this section, follows from, and is one of the most profound consequences of, the Deduction Theorem. We start by formalizing the *syntactic* notion of being able to prove an inconsistency.

Definition (Consistency)**.** A set of formulas S is said to be (syntactically) **inconsistent** with respect to a set of inference rules $\mathcal{R}$ if there exists some formula ϕ such that both ϕ and '$\sim\phi$' can be proven from the assumptions S via $\mathcal{R}$. A set of formulas that is not inconsistent with respect to $\mathcal{R}$ is said to be **consistent** with respect to $\mathcal{R}$.

You will first show that once one proves an inconsistency (prove a formula ϕ as well as prove its negation '$\sim\phi$') from some set of inference rules, it turns out that one can actually prove literally anything from the same set of inference rules, as long as one may also use both MP and the following (sound) axiom.

I2: '(~p→(p→q))'

propositions/axiomatic_systems.py

```
# Axiomatic inference rules for not (and implies)
                          ⋮
#: Implication introduction (left)
I2 = InferenceRule([], Formula.parse('(~p->(p->q))'))
```

Task 6. Implement the missing code for the function `proof_from_inconsistency(proof_of_affirmation, proof_of_negation, conclusion)`. This function takes a proof of some formula ϕ and a proof of its negation '$\sim\phi$', both via the same set of inference rules and from the same set of assumptions (implying that these assumptions are inconsistent with respect to the used set of inference rules), and also takes some arbitrary `conclusion`. The function returns a proof of `conclusion` from the same assumptions via the same inference rules as well as MP and I2.

propositions/deduction.py

```
def proof_from_inconsistency(proof_of_affirmation: Proof,
                             proof_of_negation: Proof, conclusion: Formula) -> \
        Proof:
    """Combines the given proofs of a formula `affirmation` and its negation
    '~`affirmation`' into a proof of the given formula.

    Parameters:
        proof_of_affirmation: valid proof of `affirmation`.
        proof_of_negation: valid proof of '~`affirmation`' from the same
            assumptions and inference rules of `proof_of_affirmation`.

    Returns:
        A valid proof of `conclusion` from the same assumptions as the given
        proofs, via the same inference rules as the given proofs and in addition
        `MP` and `I2`.
    """
    assert proof_of_affirmation.is_valid()
    assert proof_of_negation.is_valid()
    assert proof_of_affirmation.statement.assumptions == \
           proof_of_negation.statement.assumptions
    assert Formula('~', proof_of_affirmation.statement.conclusion) == \
           proof_of_negation.statement.conclusion
    assert proof_of_affirmation.rules == proof_of_negation.rules
    # Task 5.6
```

Hint: The function `combine_proofs()` that you implemented in Task 3 can ease the usage of the axiom I2.

From your solution to Task 6 it follows, in particular, that once any inconsistency (some ϕ and '$\sim\phi$') is proven from some set of assumptions, any other inconsistency (any other ψ and '$\sim\psi$') can also be proven. One particularly simple choice of an inconsistency is to prove a contradiction that is the negation of (a specialization of) some axiom (e.g., '$\sim(p\to p)$') since the axiom can certainly be proven. Therefore, your solution to Task 6 proves the following simple lemma.

Lemma. *Let $\mathcal{R}$ be an axiomatic system that contains MP, I0, and I2. For any set A of formulas, the following are equivalent:*

1. *A is inconsistent with respect to $\mathcal{R}$.*

2. A can prove every formula ψ via $\mathcal{R}$.

3. A can prove '~(p→p)' via $\mathcal{R}$.

We are now ready to formally justify the notion of a proof by contradiction. To do so, we will allow ourselves to use also the following additional (sound) axiom.

N: '((~q→~p)→(p→q))'

propositions/axiomatic_systems.py

```
# Axiomatic inference rules for not (and implies)
                          ⋮
#: Converse contraposition
N = InferenceRule([], Formula.parse('((~q->~p)->(p->q))'))
```

Task 7. Implement the missing code for the function `prove_by_contradiction(proof)`, which takes a proof of the contradiction '~(p→p)' from some assumptions as well as from the negation '~ϕ' of some formula, and returns a proof of the formula ϕ from these assumptions (without '~ϕ') via the same inference rules as well as MP, I0, I1, D, and N.

propositions/deduction.py

```
def prove_by_contradiction(proof: Proof) -> Proof:
    """Converts the given proof of '~(p->p)', the last assumption of which is an
    assumption '~`formula`', into a proof of `formula` from the same assumptions
    except '~`formula`'.

    Parameters:
        proof: valid proof of '~(p->p)' to convert, the last assumption of which
            is of the form '~`formula`', via some set of inference rules all of
            of which have no assumptions except perhaps `MP`.

    Return:
        A valid proof of `formula` from the same assumptions as the given proof
        except the last one, via the same inference rules as the given proof and
        in addition `MP`, `I0`, `I1`, `D`, and `N`.
    """
    assert proof.is_valid()
    assert proof.statement.conclusion == Formula.parse('~(p->p)')
    assert len(proof.statement.assumptions) > 0
    assert proof.statement.assumptions[-1].root == '~'
    for rule in proof.rules:
        assert rule == MP or len(rule.assumptions) == 0
    # Task 5.7
```

Hint: Use the Deduction Theorem (the function `remove_assumption(proof)`) and then the axiom N.

Your solutions to Tasks 6 and 7 indeed prove that a proof by contradiction is a sound proof strategy: if from ϕ you manage to prove an inconsistency[2] — both ψ and '~ψ', then

[2]Indeed the concept of a "proof by contradiction" would have perhaps been more aptly named in this book "proof by inconsistency," referencing the syntactic concept of an inconsistency rather than the semantic concept of a contradiction. The reason that it was so named is that the term contradiction is often (somewhat confusingly, at least from the point of view of propositional logic) used both in the semantic sense to reference a contradiction as defined in this book *and* in the syntactic sense to reference an inconsistency as defined in this book, and indeed the **Tautology Theorem** that we will prove in the next chapter implies that these are tightly related in that any contradiction can also be used to prove anything, much like we have shown for any inconsistency in this chapter (and we will use this when formalizing the notion of proof by contradiction for **predicate logic** in Chapter 11).

by Task 6 you can construct a proof of '~(p→p)' from ϕ, and so by Task 7, '~ϕ' can be proven. This indeed proves the soundness of proofs by contradiction, which is one of the most profound corollaries of the Deduction Theorem. What about the other direction? If we can prove ϕ from A, does this imply that $A \cup \{\text{'}\sim\phi\text{'}\}$ is inconsistent? This turns out to trivially hold since if $A \vdash \phi$, then certainly $A \cup \{\text{'}\sim\phi\text{'}\} \vdash \phi$, but also certainly $A \cup \{\text{'}\sim\phi\text{'}\} \vdash \text{'}\sim\phi\text{'}$. So we have proven not only that proof by contradiction is a sound proof strategy, but in fact that anything that can be proven can be proven via a proof by contradiction.

Theorem (Soundness of Proof by Contradiction)**.** *Let $\mathcal{R}$ be a set of inference rules that includes MP, I1, D, and N.*[3] *Let A be an arbitrary set of formulas, and let ϕ be an additional formula. Then $A \cup \{\text{'}\sim\phi\text{'}\}$ is inconsistent with respect to $\mathcal{R}$ if any only if $A \vdash_{\mathcal{R}} \phi$.*

[3]While we have actually proven this only for a set of inference rules that additionally includes also I0 and I2, you have already shown that I0 can be proven from MP, I1, and D, and we will see in the next chapter that also I2 can be proven from MP, I1, and D, so by the Lemma Theorem we already phrase the theorem accordingly.