Formal Proofs in PAL

Ryan Culpepper

Spring 2025

In this class (CS220, Spring 2025), we will use a little programming language called *Proof Assembly Language (PAL)* to learn about *formal proofs* in logic. As the name suggests, PAL is a very low-level proof language that requires every step of logical reasoning to be made explicit and justified.

Here is an example PAL program:

```
Axiom 1: Mon implies CS220
Axiom 2: Wed implies CS220
Axiom 3: CS220 implies Happy

Theorem: Mon implies Happy
1 Block for ImpliesIntro
  1.1 Assume Mon
  1.2 Want Happy
  1.3 Derive CS220            by ImpliesElim on Axiom 2, #1.1
  1.4 Derive Happy            by ImpliesElim on Axiom 3, #1.3
2 Derive Mon implies Happy    by ImpliesIntro on #1
QED
```

The example starts by declaring three *axioms*. Then a `Theorem` declaration states the goal of the proof. The *proof* itself is structured as an enumerated list that may contain nested enumerated lists. The final line of the proof derives the proposition that was stated as the goal. `QED` marks the end of the proof.

1 The Structure of a PAL Program

A PAL program is a sequence of *lines*. Each source line of the program must contain a complete declaration or proof line. That is, unlike most programming languages, PAL does not allow statements to be spread over multiple source lines. There is one exception: a `Derive` statement can be split by inserting a newline before the word `by` which starts the justification.

A comments starts with `//`, and it consists of the rest of the source line, up to but not including the newline. Comments are removed before the program is processed.

A PAL ⟨*program*⟩ consists of zero or more ⟨*declaration*⟩s, an optional ⟨*theorem*⟩ statement, zero or more ⟨*proof line*⟩s, and an optional `QED` at the end. The elements of the program must occur in that order, and `QED` is only allowed if there was a `Theorem` declared before the proof. A program is *complete* if it contains a `Theorem` declaration and ends with `QED`.

⟨*program*⟩ ::= ⟨*declaration*⟩* ⟨*theorem*⟩? ⟨*proof line*⟩* `QED`?

⟨*declaration*⟩ ::= `Axiom` ⟨*index*⟩ `:` ⟨*proposition*⟩

| `Declare` ⟨*set name*⟩ `=` `{` ⟨*object constant*⟩+ `,` `...`? `}`

⟨*theorem*⟩ ::= `Theorem` `:` ⟨*proposition*⟩

⟨*proof line*⟩ ::= ⟨*line number*⟩ ⟨*statement*⟩

Each ⟨*proof line*⟩ consists of a ⟨*line number*⟩ followed by a ⟨*statement*⟩. A ⟨*line number*⟩ is either a positive integer or a dot-separated sequence of positive integers. Examples are `1`, `1.4`, and `1.4.2.5`. The number of components of a ⟨*line number*⟩ is determined by the number of *blocks* the line is within. The line number of every line within a block is an extension of the line number of the `Block` statement itself. Line numbers must increase, but skipping numbers is allowed.

1.1 Statements

⟨*statement*⟩ ::= `Derive` ⟨*proposition*⟩ `by` ⟨*justification*⟩
| `Block for` ⟨*block rule*⟩
| `Assume` ⟨*proposition*⟩
| `Let` ⟨*variable*⟩ `in` ⟨*set name*⟩
| `Want` ⟨*proposition*⟩

A ⟨*statement*⟩ is one of the following:

- "`Derive` ⟨*proposition*⟩ `by` ⟨*justification*⟩"

  This line derives a new proposition using one of the rules of inference. The rule and its arguments form the ⟨*justification*⟩. The derived proposition is *available* until the end of the block.

  The rules available are explained in the following sections.

- "`Block for` ⟨*block rule*⟩"

  Marks the start of a *block* (aka *assumption block*). The lines belonging to the block are indented, and their labels are prefixed with the block's label. (For example, if line `1.4` has the `Block` statement, then the following lines are `1.4.1`, `1.4.2`, and so on, until the end of the block.)

  The block must start with an `Assume` or `Let` statement. The ⟨*block rule*⟩ declares what rule will use the block, and each rule imposes specific requirements on the contents of the block.

  Everything within the block becomes *unavailable* once the block ends.

- "`Assume` ⟨*proposition*⟩"

  Only allowed at the beginning of a *block*. The assumption is *available* until the end of the block.

- "`Let` ⟨*variable*⟩ `in` ⟨*set name*⟩"

  Only allowed at the beginning of a *block*. The object variable named ⟨*variable*⟩ must not already be in scope, and after this line it is in scope for the rest of the block.

- "`Want` ⟨*proposition*⟩"

  This statement is essentially another kind of comment. For example, it may be useful to include a `Want` line near the beginning of a block (after the assumption) to document the immediate goal of the block. A `Want` statement does not make its proposition *available* (wanting is not having).

1.2 Propositions

⟨*proposition*⟩ ::= `forall` ⟨*variable*⟩ `in` ⟨*set name*⟩ `,` ⟨*proposition*⟩
| `exists` ⟨*variable*⟩ `in` ⟨*set name*⟩ `,` ⟨*proposition*⟩
| ⟨*proposition*⟩ `or` ⟨*proposition*⟩
| ⟨*proposition*⟩ `implies` ⟨*proposition*⟩
| ⟨*proposition*⟩ `iff` ⟨*proposition*⟩
| ⟨*proposition*⟩ `and` ⟨*proposition*⟩
| `not` ⟨*proposition*⟩
| ⟨*atomic proposition*⟩
| ⟨*predicate name*⟩ `(` ⟨*expression*⟩$^+$ `)`
| ⟨*expression*⟩ `=` ⟨*expression*⟩
| `(` ⟨*proposition*⟩ `)`

An ⟨*atomic proposition*⟩ and a ⟨*predicate name*⟩ are both written as an identifier that starts with an uppercase letter. For example: `Sunny` is an atomic proposition and `Divides` is a predicate name in the proposition `Sunny implies Divides(4, 36)`.

A ⟨*proposition*⟩ must not refer to any object variable that is not in scope.

The symbols $\forall, \exists, \in, \vee, \Rightarrow, \Leftrightarrow, \wedge, \neg$ can be used in place of `forall`, etc.

The grammar lists productions in order of decreasing precedence. For example, `not A and B or C` is parsed as `((not A) and B) or C`. The `and` and `or` connectives associate to the left; that is, `A and B and C` is parsed as `(A and B) and C`. The `implies` connective associates to the right: `A implies B implies C` is parsed as `A implies (B implies C)`. The other connectives are non-associative; for example, `A iff B iff C` is invalid syntax.

Nested quantifiers over the same set may be abbreviated to a single quantifier with a list of variables. Thus, `forall a,b,c in NN, P` is parsed as `forall a in NN, forall b in NN, forall C in NN, P`, or equivalently, as `forall a in NN, (forall b in NN, (forall c in NN, P))`.

1.3 Expressions

⟨*expression*⟩ ::= ⟨*expression*⟩ `+` ⟨*expression*⟩
| ⟨*expression*⟩ `*` ⟨*expression*⟩
| ⟨*natural number*⟩
| ⟨*object variable*⟩
| ⟨*object constant*⟩
| `(` ⟨*expression*⟩ `)`

An ⟨*object variable*⟩ is written as an identifier that starts with a lowercase letter. For example: `x`, `abc99`.

An ⟨*object constant*⟩ is written as identifier enclosed in single quotes. It is not an arbitrary string—in particular, spaces are not allowed. For example: `'Lion'`, `'Mouse'`.

An ⟨*expression*⟩ must not refer to any variables that are not in scope.

1.4 Justifications

Every `Derive` statement must have an accompanying justification:

$$
\begin{array}{rcl}
\langle \textit{justification} \rangle & ::= & \langle \textit{rule name} \rangle\ \langle \textit{justification argument} \rangle^* \\
\langle \textit{justification argument} \rangle & ::= & \texttt{on}\ \langle \textit{reference} \rangle^+ \\
& | & \texttt{with}\ \langle \textit{variable mapping} \rangle \\
\langle \textit{reference} \rangle & ::= & \langle \textit{proposition reference} \rangle\ |\ \langle \textit{block reference} \rangle \\
\langle \textit{proposition reference} \rangle & ::= & \texttt{Axiom}\ \langle \textit{index} \rangle\ |\ \texttt{\#}\langle \textit{line number} \rangle \\
\langle \textit{block reference} \rangle & ::= & \texttt{\#}\langle \textit{line number} \rangle
\end{array}
$$

A ⟨*justification*⟩ consists of a ⟨*rule name*⟩, referring to an *inference rule*, and its arguments. Inference rules are discussed in the following sections. The most common kind of argument is the ⟨*reference*⟩, which specifies how the rule's premises are satisfied:

- A ⟨*proposition reference*⟩ refers to an available *proposition*. There are two forms:
  - "`Axiom` ⟨*index*⟩" refers to the declared axiom with the given index.
  - "`#`⟨*line number*⟩" refers to the proposition that was `Derive`d or `Assume`d on the line with the given line number. That line must still be *available* at the current line—that is, it must not belong to a *block* that has ended.
- A ⟨*block reference*⟩ is written "`#`⟨*line number*⟩" and it refers to the entire *block* started by a `Block` statement on the line with the given line number.

  The `Block` statement's line must still be *available*, but the block itself must have ended. For example, in the proof above, at line `#2`, the `Block` statement on line `#1` is *available*, but the block itself has ended, so the justification is legal.

The other kind of ⟨*justification argument*⟩ is the ⟨*variable mapping*⟩. It is discussed in Section 3.

2 Rules for Propositional Logic (Direct Proof)

Inference rules for the logical connectives of propositional logic:

$$\frac{p \wedge q}{p}\ \wedge\textsc{ElimL} \qquad \frac{p \wedge q}{q}\ \wedge\textsc{ElimR} \qquad \frac{p \qquad q}{p \wedge q}\ \wedge\textsc{Intro}$$

$$\frac{p \vee q \qquad p \Rightarrow r \qquad q \Rightarrow r}{r}\ \vee\textsc{Elim} \qquad \frac{p}{p \vee q}\ \vee\textsc{IntroL} \qquad \frac{q}{p \vee q}\ \vee\textsc{IntroR}$$

$$\frac{p \Rightarrow q \qquad p}{q}\ \Rightarrow\textsc{Elim} \qquad \frac{\begin{array}{|l}\texttt{Assume}\ p \\ \quad\vdots \\ \quad q\end{array}}{p \Rightarrow q}\ \Rightarrow\textsc{Intro}$$

$$\frac{p \Leftrightarrow q}{p \Rightarrow q}\ \Leftrightarrow\textsc{ElimF} \qquad \frac{p \Leftrightarrow q}{q \Rightarrow p}\ \Leftrightarrow\textsc{ElimB} \qquad \frac{p \Rightarrow q \qquad q \Rightarrow p}{p \Leftrightarrow q}\ \Leftrightarrow\textsc{Intro}$$

The rules have the following ⟨*justification*⟩ syntax:

- "`by AndElimL on` ⟨*proposition reference*⟩"
- "`by AndElimR on` ⟨*proposition reference*⟩"
- "`by AndIntro on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"

- "`by OrElim on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"
- "`by OrIntroL on` ⟨*proposition reference*⟩"
- "`by ImpliesElim on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"
- "`by ImpliesIntro on` ⟨*block reference*⟩"

  The block must start with an `Assume` statement.
- "`by IffElimF on` ⟨*proposition reference*⟩"

  This rule extracts the "forward" implication.
- "`by IffElimB on` ⟨*proposition reference*⟩"

  This rule extracts the "backward" implication.
- "`by IffIntro on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"

The rule names may also be written with symbolic version of the connective name. For example, `AndIntro` may be written as `∧Intro`, and `ExistsElim` may be written as `∃Elim`, etc.

2.1 Example: Conjunction

Axioms:

1. Contains(water, hydrogen) ∧ Contains(water, oxygen)
2. Contains(rust, oxygen) ∧ Contains(rust, iron)

Prove: Contains(rust, oxygen) ∧ Contains(water, oxygen)

```
Axiom 1: Contains('water', 'hydrogen') and Contains('water', 'oxygen')
Axiom 2: Contains('rust', 'oxygen') and Contains('rust', 'iron')

Theorem: Contains('rust', 'oxygen') and Contains('water', 'oxygen')
1 Derive Contains('rust', 'oxygen') by AndElimL on Axiom 2
2 Derive Contains('water', 'oxygen') by AndElimR on Axiom 1
3 Derive Contains('rust', 'oxygen') and Contains('water', 'oxygen')
  by AndIntro on #1, #2
QED
```

2.2 Example: Implication

Axioms:

1. *Mon* ⇒ *CS220*
2. *Wed* ⇒ *CS220*
3. *CS220* ⇒ *Happy*

Prove: *Mon* ⇒ *Happy*

```
Axiom 1: Mon implies CS220
Axiom 2: Wed implies CS220
Axiom 3: CS220 implies Happy

Theorem: Mon implies Happy
1 Block for ImpliesIntro
  1.1 Assume Mon
  1.2 Want Happy
  1.3 Derive CS220 by ImpliesElim on Axiom 1, #1.1
```

```
  1.4 Derive Happy by ImpliesElim on Axiom 3, #1.3
2 Derive Mon implies Happy by ImpliesIntro on #1
QED
```

2.3 Example: Disjunction

Axioms:

1. $Sun \Rightarrow (Bike \wedge Garden)$
2. $Rain \Rightarrow Clean$

Prove: $(Rain \vee Sun) \Rightarrow (Garden \vee Clean)$

```
Axiom 1: Sun implies (Bike and Garden)
Axiom 2: Rain implies Clean

Theorem: (Rain or Sun) implies (Garden or Clean)
1 Block for ImpliesIntro
  1.1 Assume Rain or Sun
  1.2 Want Garden or Clean
  // want Rain implies (Garden or Clean)
  1.3 Block for ImpliesIntro
    1.3.1 Assume Rain
    1.3.2 Want Garden or Clean
    1.3.3 Derive Clean by ImpliesElim on Axiom 2, #1.3.1
    1.3.4 Derive Garden or Clean by OrIntroR on #1.3.3
  1.4 Derive Rain implies (Garden or Clean) by ImpliesIntro on #1.3
  // want Sun implies (Garden or Clean)
  1.5 Block for ImpliesIntro
    1.5.1 Assume Sun
    1.5.2 Want Garden or Clean
    1.5.3 Derive Bike and Garden by ImpliesElim on Axiom 1, #1.5.1
    1.5.4 Derive Garden by AndElimR on #1.5.3
    1.5.5 Derive Garden or Clean by OrIntroL on #1.5.4
  1.6 Derive Sun implies (Garden or Clean) by ImpliesIntro on #1.5
  1.7 Derive Garden or Clean by OrElim on #1.1, #1.4, #1.6
2 Derive (Rain or Sun) implies (Garden or Clean) by ImpliesIntro on #1
QED
```

It would also be valid to derive the two "case handlers" earlier, outside of the $Rain \vee Sun$ assumption:

```
Axiom 1: Sun implies (Bike and Garden)
Axiom 2: Rain implies Clean

Theorem: (Rain or Sun) implies (Garden or Clean)
// want Rain implies (Garden or Clean)
1 Block for ImpliesIntro
  1.1 Assume Rain
  1.2 Want Garden or Clean
  1.3 Derive Clean by ImpliesElim on Axiom 2, #1.1
  1.4 Derive Garden or Clean by OrIntroR on #1.3
2 Derive Rain implies (Garden or Clean) by ImpliesIntro on #1
// want Sun implies (Garden or Clean)
3 Block for ImpliesIntro
```

```
  3.1 Assume Sun
  3.2 Want Garden or Clean
  3.3 Derive Bike and Garden by ImpliesElim on Axiom 1, #3.1
  3.4 Derive Garden by AndElimR on #3.3
  3.5 Derive Garden or Clean by OrIntroL on #3.4
4 Derive Sun implies (Garden or Clean) by ImpliesIntro on #3
5 Block for ImpliesIntro
  5.1 Assume Rain or Sun
  5.2 Want Garden or Clean
  5.3 Derive Garden or Clean by OrElim on #5.1, #2, #4
6 Derive (Rain or Sun) implies (Garden or Clean) by ImpliesIntro on #5
QED
```

3 Rules for Predicate Logic

Inference rules for the quantifiers of predicate logic:

$$\frac{\forall x \in A,\ P(x) \qquad a \in A}{P(a)}\ \forall\textsc{Elim} \qquad\qquad \frac{\begin{array}{|l}\texttt{Let } x \in A \\ \vdots \\ P(x)\end{array}}{\forall x \in A,\ P(x)}\ \forall\textsc{Intro}$$

$$\frac{\exists x \in A,\ P(x) \qquad \begin{array}{|l}\texttt{Let } y \in A \\ \texttt{Assume } P(y) \\ \vdots \\ q\end{array}}{q}\ \exists\textsc{Elim} \qquad\qquad \frac{P(a) \qquad a \in A}{\exists x \in A,\ P(x)}\ \exists\textsc{Intro}$$

The rules have the following ⟨*justification*⟩ syntax:

- "`by ForAllElim on` ⟨*proposition reference*⟩ `with` ⟨*variable mapping*⟩"
- "`by ForAllIntro on` ⟨*block reference*⟩"

  The block must start with a `Let` statement. The variable introduced by the `Let` statement must not already be in scope.
- "`by ExistsElim on` ⟨*proposition reference*⟩`,` ⟨*block reference*⟩"

  The block must start with a `Let` statement followed by an `Assume` statement, and the assumption must consist of the existential proposition's body with the existential variable replaced by the new variable. The new variable must not already be in scope.
- "`by ExistsIntro on` ⟨*proposition reference*⟩ `with` ⟨*variable mapping*⟩"

The `ForAllElim` and `ExistsIntro` rules require a new kind of ⟨*justification argument*⟩:

- A ⟨*variable mapping*⟩ is written after the word `with` using the following syntax:

  ⟨*variable mapping*⟩ ::= ⟨*variable*⟩ `:=` ⟨*expression*⟩

  The variable on the left refers to the variable belonging to a *quantifier* (whether the quantifier is being introduced or eliminated), and the expression on the right refers to the expression that replaces the variable (for ∀Elim) or the witness expression that the variable hides (for ∃Intro).

  The "gets" symbol, `:=`, can also be written as ↦ ("maps-to").

3.1 Variable Management

Rules for variables:

- No expression—whether in an axiom, assumption or other statement, or variable mapping—may have a variable that is not in scope.
- No `Let` statement may introduce a variable that is already in scope.
- Substitution (for ∀Elim and ∃Intro) must avoid *variable capture*.

Variable capture occurs when an expression containing a free variable is substituted into a proposition into a position where the same variable is bound. For example, consider the following (true) proposition:

$$\forall x \in \mathbb{N},\ \exists y \in \mathbb{N},\ y > x$$

If we consider the body of this universal proposition as an open statement P, then we would say:

$$P(x)\ =\ \exists y \in \mathbb{N},\ y > x$$

Consider the following "naive" substitutions (for example, for ∀Elim):

- $P(5)\ =\ \exists y \in \mathbb{N},\ y > 5$ — This is fine.
- $P(3z+2)\ =\ \exists y \in \mathbb{N},\ y > 3z+2$ — This is fine, if $z \in \mathbb{N}$ is in scope where we perform the instantiation.
- $P(y)\ =\ \exists y \in \mathbb{N},\ y > y$ — **Wrong!** The y in the resulting proposition no longer refers to the y in scope where we decided to perform the instantiation; rather, it has been *captured* by the ∃ quantifier. And the resulting proposition is false!

One solution is to use *capture-avoiding substitution*, which renames bound variables if necessary:

- $P(y)\ =\ \exists z \in \mathbb{N},\ z > y$ — Note, the ∃-bound y from the original proposition has been renamed to z.

This is okay, because the names of quantifier-bound variables does not matter, as long as they are used in the same way. For example, $\exists z \in \mathbb{N},\ z > y$ means exactly the same thing as $\exists x \in \mathbb{N},\ x > y$ and exactly the same thing as $\exists a \in \mathbb{N},\ a > y$. All three assert the existence of some natural number greater than y, where y must be a variable currently in scope.

(Renaming bound variables is sometimes called "α-renaming" or "α-conversion". Terms equivalent except for the choice of bound variable names are called "α-equivalent". These terms come from λ-calculus.)

PAL allows the proposition in a `Derive` statement to be any proposition α-equivalent to the proposition derived by the given justification. For example, the following PAL snippet is allowed:

```
Axiom 1: forall x in NN, exists y in NN, y > x
1 Derive exists n in NN, n > 5 by ForAllElim on Axiom 1 with x := 5
```

because $\forall y \in \mathbb{N},\ y > 5$ is α-equivalent to $\forall n \in \mathbb{N},\ n > 5$.

3.2 Example: Universals

Let $A = \{\mathsf{Mouse}, \mathsf{Lion}, \dots\}$.

Axioms:

1. $\text{Small}(\mathsf{Mouse})$
2. $\text{Brave}(\mathsf{Lion})$
3. $\forall a, b \in A,\ (\text{Small}(a) \wedge \text{Brave}(b)) \Rightarrow \text{Fears}(a, b)$

Prove: $\text{Fears}(\mathsf{Mouse}, \mathsf{Lion})$

```
Declare A = {'Mouse', 'Lion', ...}
Axiom 1: Small('Mouse')
Axiom 2: Brave('Lion')
Axiom 3: forall a, b in A, (Small(a) and Brave(b)) implies Fears(a,b)

Theorem: Fears('Mouse', 'Lion')
1 Derive forall b in A, (Small('Mouse') and Brave(b)) implies Fears('Mouse', b)
  by ForAllElim on Axiom 3 with a := 'Mouse'
2 Derive (Small('Mouse') and Brave('Lion')) implies Fears('Mouse', 'Lion')
  by ForAllElim on #1 with b := 'Lion'
3 Derive Small('Mouse') and Brave('Lion') by AndIntro on Axiom 1, Axiom 2
4 Derive Fears('Mouse', 'Lion') by ImpliesElim on #2, #3
QED
```

3.3 Example: Existentials

Axioms:

1. $\forall n \in \mathbb{N},\ \text{Even}(n) \Leftrightarrow (\exists k \in \mathbb{N},\ n = 2k)$
2. $\forall n \in \mathbb{N},\ \text{Odd}(n) \Leftrightarrow (\exists k \in \mathbb{N},\ n = 2k + 1)$

Prove: $\forall n \in \mathbb{N},\ \text{Odd}(n) \ \Rightarrow\ \text{Even}(n + 1)$

```
Axiom 1: forall n in NN, Even(n) iff (exists k in NN, n = 2*k)
Axiom 2: forall n in NN, Odd(n) iff (exists k in NN, n = 2*k + 1)

Theorem: forall m in NN, Odd(m) implies Even(m+1)
1 Block for ForAllIntro
  1.1 Let m in NN
  1.2 Want Odd(m) implies Even(m+1)
  1.3 Block for ImpliesIntro
    1.3.1 Assume Odd(m)
    1.3.2 Want Even(m+1)
    1.3.3 Derive Odd(m) iff (exists k in NN, m = 2*k + 1)
          by ForAllElim on Axiom 2 with n := m
    1.3.4 Derive Odd(m) implies (exists k in NN, m = 2*k + 1)
          by IffElimF on #1.3.3
    1.3.5 Derive exists k in NN, m = 2*k + 1
          by ImpliesElim on #1.3.4, #1.3.1
    1.3.6 Block for ExistsElim
      1.3.6.1 Let kk in NN
      1.3.6.2 Assume m = 2*kk + 1
      1.3.6.3 Derive m+1 = 2*(kk + 1) by Algebra on #1.3.6.2
      1.3.6.4 Derive exists k in NN, m+1 = 2*k
              by ExistsIntro on #1.3.6.3 with k := kk+1
    1.3.7 Derive exists k in NN, m+1 = 2*k
          by ExistsElim on #1.3.5, #1.3.6
    1.3.8 Derive Even(m+1) iff (exists k in NN, m+1 = 2*k)
          by ForAllElim on Axiom 1 with n := m+1
    1.3.9 Derive (exists k in NN, m+1 = 2*k) implies Even(m+1)
          by IffElimB on #1.3.8
    1.3.10 Derive Even(m+1) by ImpliesElim on #1.3.9, #1.3.7
  1.4 Derive Odd(m) implies Even(m+1) by ImpliesIntro on #1.3
2 Derive forall m in NN, Odd(m) implies Even(m+1) by ForAllIntro on #1
```

```
QED
```

4 Rules for Propositional Logic (Indirect Proof)

Inference rules for the logical connectives of propositional logic:

$$\frac{p \Rightarrow q \qquad \neg q}{\neg p}\ \textsc{Modus Tollens} \qquad\qquad \frac{p \vee q \qquad \neg p}{q}\ \textsc{Disjunctive Syllogism}$$

$$\frac{\begin{array}{|l}\texttt{Assume } p \\ \quad\vdots \\ \quad q \wedge \neg q\end{array}}{\neg p}\ \textsc{Contradiction}$$

The rules have the following ⟨*justification*⟩ syntax:

- "`by ModusTollens on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"
- "`by DisjunctiveSyllogism on` ⟨*proposition reference*⟩`,` ⟨*proposition reference*⟩"
- "`by Contradiction on` ⟨*block reference*⟩"

  The block must start with an `Assume` statement.

4.1 Example: Indirect Proof

Axioms:

1. *Stranger* ∨ *Household*
2. *Stranger* ⇒ *Bark*

Prove: ¬*bark* ⇒ *household*

```
Axiom 1: Stranger or Household
Axiom 2: Stranger implies Bark

Theorem: not Bark implies Household
1 Block for ImpliesIntro
  1.1 Assume not Bark
  1.2 Want Household
  1.3 Derive not Stranger by ModusTollens on Axiom 2, #1.1
  1.4 Derive Household by DisjunctiveSyllogism on Axiom 1, #1.3
2 Derive not Bark implies Household by ImpliesIntro on #1
QED
```

4.2 Example: Contradiction

Prove: ¬A ⇒ ¬(A ∧ B)

```
Theorem: not A implies not (A and B)
1 Block for ImpliesIntro
  1.1 Assume not A
  1.2 Want not (A and B)
  1.3 Block for Contradiction
    1.3.1 Assume A and B
    1.3.2 Want A and not A
```

```
    1.3.3 Derive A by AndElimL on #1.3.1
    1.3.4 Derive A and not A by AndIntro on #1.3.3, #1.1
  1.4 Derive not (A and B) by Contradiction on #1.3
2 Derive not A implies not (A and B) by ImpliesIntro on #1
QED
```

5 Additional Inference Rules

Additional inference rules:

$$\frac{p}{p}\ \textsc{Repeat} \qquad\qquad \frac{e_1 = e_1' \quad \ldots \quad e_n = e_n'}{e = e'}\ \textsc{Algebra}$$

The rules have the following ⟨*justification*⟩ syntax:

- "`by Repeat on` ⟨*proposition reference*⟩"

  This rule can be useful for "deriving" a proposition by simply restating an assumption. It can also be used to restate a proposition with different bound variables to avoid variable capture.

- "`by Algebra on` ⟨*proposition reference*⟩`*`"

  The Algebra rule may only derive equations, and it takes zero or more equations as arguments. The derived equation must be true whenever all of the premises are true.

5.1 Example: Repeat

Prove: $A \Rightarrow A$

```
Theorem: A implies A
1 Block for ImpliesIntro
  1.1 Assume A
  1.2 Derive A by Repeat on #1.1
2 Derive A implies A by ImpliesIntro on #1
QED
```

5.2 Example: Algebra

Prove: $\forall n \in \mathbb{N},\ 2n = n + n$

```
Theorem: forall n in NN, 2*n = n + n
1 Block for ForAllIntro
  1.1 Let n in NN
  1.2 Derive 2*n = n*n by Algebra
2 Derive forall n in NN, 2*n = n + n by ForAllIntro
QED
```

6 Relaxed Proof Structure

Many axioms, lemmas, and theorems have the following structure:[1]

$$\forall a_1 \in A_1, a_2 \in A_2, \ldots, \forall a_m \in A_n,\ p_1 \Rightarrow p_2 \Rightarrow \ldots \Rightarrow p_n \Rightarrow q$$

That is, there are one or more $\forall$ quantifiers binding the variables $a_1, \ldots, a_m$, followed by one or more premises $(p_1, \ldots, p_n)$, and then a conclusion (q). For example,

$$\forall a \in \mathbb{N},\ \forall b \in \mathbb{N},\ \forall c \in \mathbb{N},\ \forall d \in \mathbb{N},\ a \leq b \Rightarrow c \leq d \Rightarrow a + c \leq b + d$$

[1] Note $\Rightarrow$ is not associative; $A \Rightarrow B \Rightarrow C$ means $A \Rightarrow (B \Rightarrow C)$.

Suppose that we want to use that lemma to prove $\forall m, n \in \mathbb{N},\ m \leq n \Rightarrow m + 1 \leq n + 2$. Here is the proof according to the strict proof rules:

```
Axiom 1: forall a,b,c,d in NN, a <= b implies c <= d implies a+c <= b+d

Theorem: forall m,n in NN, m <= n implies m+1 <= n+2
1 Block for ForAllIntro
  1.1 Let m in NN
  1.2 Block for ForAllIntro
    1.2.1 Let n in NN
    1.2.2 Block for ImpliesIntro
      1.2.2.1 Assume m <= n
      1.2.2.2 Want m+1 <= n+2
      1.2.2.3 Derive 1 <= 2 by Algebra
      1.2.2.4 Derive forall b,c,d in NN, m <= b implies c <= d implies m+c <= b+d
              by ForAllElim on Axiom 1 with a := m
      1.2.2.5 Derive forall c,d in NN, m <= n implies c <= d implies m+c <= n+d
              by ForAllElim on #1.2.2.4 with b := n
      1.2.2.6 Derive forall d in NN, m <= n implies 1 <= d implies m+1 <= n+d
              by ForAllElim on #1.2.2.5 with c := 1
      1.2.2.7 Derive m <= n implies 1 <= 2 implies m+1 <= n+2
              by ForAllElim on #1.2.2.6 with d := 2
      1.2.2.8 Derive 1 <= 2 implies m+1 <= n+2
              by ImpliesElim on #1.2.2.7, #1.2.2.1
      1.2.2.9 Derive m+1 <= n+2
              by ImpliesElim on #1.2.2.8, #1.2.2.3
    1.2.3 Derive m <= n implies m+1 <= n+2 by ImpliesIntro on #1.2.2
  1.3 Derive forall n in NN, m <= n implies m+1 <= n+2 by ForAllIntro on #1.2
2 Derive forall m,n in NN, m <= n implies m+1 <= n+2 by ForAllIntro on #1
QED
```

6.0.1 Relaxed ∀⇒-Elimination

You may combine multiple ∀Elim steps followed by multiple ⇒Elim steps into a single step. Write the justification as

"`by` ⟨*proposition reference*⟩ `with` ⟨*multi-variable mapping*⟩ `on` ⟨*proposition reference*⟩$^+$"

where ⟨*multi-variable mapping*⟩ ::= ⟨*variable*⟩$^+$ `:=` ⟨*expression*⟩$^+$

The first ⟨*proposition reference*⟩ refers to the original universal proposition, the ⟨*multi-variable mapping*⟩'s comma-separated ⟨*variable*⟩s and ⟨*expression*⟩s are the ∀-bound variables and their replacement expressions (in order), and the remaining ⟨*proposition reference*⟩s are the premises of the implications (in order).

With the relaxed elimination rule, the previous proof simplifies to the following:

```
Axiom 1: forall a,b,c,d in NN, a <= b implies c <= d implies a+c <= b+d

Theorem: forall m,n in NN, m <= n implies m+1 <= n+2
1 Block for ForAllIntro
  1.1 Let m in NN
  1.2 Block for ForAllIntro
    1.2.1 Let n in NN
    1.2.2 Block for ImpliesIntro
      1.2.2.1 Assume m <= n
      1.2.2.2 Want m+1 <= n+2
```

```
      1.2.2.3 Derive 1 <= 2 by Algebra
      1.2.2.4 Derive m+1 <= n+2
              by Axiom 1 with a,b,c,d := m,n,1,2 on #1.2.2.1, #1.2.2.3
    1.2.3 Derive m <= n implies m+1 <= n+2 by ImpliesIntro on #1.2.2
  1.3 Derive forall n in NN, m <= n implies m+1 <= n+2 by ForAllIntro on #1.2
2 Derive forall m,n in NN, m <= n implies m+1 <= n+2 by ForAllIntro on #1
QED
```

6.0.2 Relaxed ∀⇒-Introduction

You may combine nested blocks that only perform ∀Intro steps (outermost) and ⇒Intro steps (innermost) into a single block. The combined block will have a sequence of `Let` statements followed by a sequence of `Assume` statements. The justification of the `Derive` statement after the block should be "`by Intro on` ⟨*block reference*⟩".

With the relaxed introduction rule, the previous proof further simplifies to the following:

```
Axiom 1: forall a,b,c,d in NN, a <= b implies c <= d implies a+c <= b+d

Theorem: forall m,n in NN, m <= n implies m+1 <= n+2
1 Block for Intro
  1.1 Let m,n in NN
  1.2 Assume m <= n
  1.3 Want m+1 <= n+2
  1.4 Derive 1 <= 2 by Algebra
  1.5 Derive m+1 <= n+2
      by Axiom 1 with a,b,c,d := m,n,1,2 on #1.2, #1.4
2 Derive forall m,n in NN, m <= n implies m+1 <= n+2 by Intro on #1
QED
```

6.0.3 Relaxed ∀⇔⇒-Elimination

Another common structure, especially for definitions, is the following:

$$\forall a_1 \in A_1, a_2 \in A_2, \ldots, \forall a_n \in A_n,\ p \iff q$$

You can abbreviate a sequence of ∀Elim steps, a ⇔ElimF or ⇔ElimB step, and an ⇒Elim step into a single step. Use the same justification notation as relaxed ∀⇒-elimination, but write `forward` or `backward` between the "with" and "on" clauses.