This document consists of two parts: information on how to run the programme, and a guide on how to use it.

Running the PRODLOG Proof Checker

The entry point to the programme is GUIView.java. To run this file, the following external JAR files need to be added to the program's module path:

- javafx-swt.jar
- javafx.base.jar
- javafx.controls.jar
- javafx.fxml.jar
- javafx.graphics.jar
- javafx.media.jar
- javafx.swing.jar
- javafx.web.jar

The JDK 11 library was used as the specific JRE for this project - it is strongly recommended that the code be run with JDK11.

The following VM arguments need to be added to the run configurations:

- --module-path (location of JavaFX JAR files in memory)
- --add-modules=javafx.base
- --add-modules=javafx.controls
- --add-modules=javafx.fxml
- --add-modules=javafx.graphics
- --add-modules=javafx.media
- --add-modules=javafx.swing
- --add-modules=javafx.web

To run the JUnit test classes, the JUnit 5 library should be configured in the class path.

Furthermore, to ensure correct outputs, run with a UTF-8 character set.

If in doubt, open and run the code in Eclipse.

Quick-Start User Guide for PRODLOG

1. Introduction to Using PRODLOG

The purpose of this section is to help the user to begin to interact with the PRODLOG system for practicing Fitch-style natural deduction. It contains four parts. The first is this general introduction. The next two introduce the languages used to interact with the system. The final section lists the inference rules stored within the system.

To interact with the system, type your proof into the large text area on the lefthand side of the display with each new line of proof occupying a new line in the input field. Every time you press enter, the system will compile the input so far, and attempt to turn it into a Fitch-style proof, which is output on the right.

Pressing on the "Propositional Logic" and "Predicate Logic" tabs at the top of the page will change the way that you can interact with the system, and the way that the system behaves. When set to Propositional Logic, use the PROPLOG language specified immediately below in section 2. When set to Predicate Logic, use the PREDLOG language specified further down in section 3. The languages closely resemble the syntax of propositional and predicate logic, respectively. To help improve the readability of your proofs, inline comments are permitted and denoted using '//'. Multiline comments are also allowed, denoted using '/* *your comment* */'.

2. Vocabulary, Syntax, and Semantics of PROPLOG

The vocabulary of PROPLOG can be split into three broad categories. First is the vocabulary and semantics for formulating propositions:

Type of Proposition	**Word of Vocabulary**	**Represents**
Binary	AND	Conjunction
	OR	Disjunction
	IFTHEN	Conditional
	IFF	Biconditional
Unary	NOT	Negation
Atomic	One capitalised alphabetic letter, with or without trailing numbers	Atomic proposition
	ABSURD	Logical absurdity

The "represents" column describes the concept of propositional logic represented by the word of PROPLOG. The vocabulary is grouped by type based on whether the lexeme joins two sentences of PROPLOG (binary), ranges over one sentence of PROPLOG (unary), or is an atomic sentence of the language (atomic). The significance of this will soon be described when considering the syntax of the language.

The second category of the PROPLOG vocabulary is keywords:

Word of Vocabulary	**Represents**
#PREMISE	Assert a premise
#CONCLUDE	Assert a conclusion
#ASSUME	Assert an assumption
#ENDASSUME	Close the outermost assumption
#DERIVE	Assert a new derivation

Again, the concept from propositional logic represented by each word is included in the "Represents" category.

The third is symbols:

Symbol	Represents
(	Open bracket
)	Close bracket

The syntax of PROPLOG is simple enough and builds on this separation of the vocabulary into categories. We first draw a distinction between propositions and sentences of the language. A sentence of the language can be defined as a valid input of the system. A proposition of the language can be defined as something which, if paired with a keyword, would constitute a sentence of the language. In practice, this is what each line of your input should contain.

Therefore, we can recursively define propositions of the language as follows:

If ***A*** is an atomic proposition, then ***A*** is a proposition of PROPLOG.

If ***B*** is a proposition of PROPLOG and ***B*** *is not* enclosed in brackets, then NOT(***B***) is a proposition of PROPLOG.

If ***B*** is a proposition of PROPLOG and ***B*** *is* enclosed in brackets, then NOT***B*** is a proposition of PROPLOG.

If ***C*** and ***D*** are both propositions of PROPLOG, then:

- (***C*** AND ***D***)
- (***C*** OR ***D***)
- (***C*** IFTHEN ***D***)
- (***C*** IFF ***D***)

Are all propositions of PROPLOG.

Nothing else is a proposition of PROPLOG.

Based on this definition of a proposition, we can define a sentence of PROPLOG as follows:

If ***E*** is a proposition of PROPLOG, then:

- #PREMISE ***E***
- #CONCLUDE ***E***
- #ASSUME ***E***
- #DERIVE ***E***

Are all sentences of PROPLOG.

#ENDASSUME is a sentence of PROPLOG.

Nothing else is a sentence of PROPLOG.

3. Vocabulary, Syntax, and Semantics of PREDLOG

PREDLOG is to predicate logic what PROPLOG is to propositional logic. Quantifiers are introduced, terms are introduced, and the basic unit of the language shifts from being an atomic proposition to an atomic formula. The vocabulary and semantics of PREDLOG are as follows:

Elements of the vocabulary with type term:

Type of Term	Word of Vocabulary	Represents
Constant	One lowercase alphabetic letter from a-r inclusive, with or without trailing numbers	Constant
Variable	One lowercase alphabetic letter from s-z inclusive, with or without trailing numbers	Variable

Elements of the vocabulary with type symbol:

Symbol	Represents
(	Open bracket
)	Close bracket
=	Identity symbol

Elements of the vocabulary with type proposition:

Type of Proposition	Word of Vocabulary	Represents
Binary	AND	Conjunction
	OR	Disjunction
	IFTHEN	Conditional
	IFF	Biconditional
Unary	NOT	Negation
Atomic Formula	One capitalised alphabetic letter, with or without trailing numbers, followed by a parenthesised, nonempty set of terms	Predicate-Term(s)
	One term, followed by the identity symbol, followed by another term	Identity formula
	ABSURD	Logical absurdity
Quantifier	FORALL	Universal quantification
	FORSOME	Existential quantification

Note here that the capitalised letter of the predicate-term combination is referred to as a predicate (e.g. the 'A' in 'A(gyz)'). A predicate alone is not a word of the language.

Finally, the keyword set (this is the same as for PROPLOG):

Word of Vocabulary	Represents
#PREMISE	Assert a premise
#CONCLUDE	Assert a conclusion
#ASSUME	Assert an assumption
#ENDASSUME	Close the outermost assumption
#DERIVE	Assert a new derivation

A slight change in terms is used to define a sentence of PREDLOG, since a variable must be bound by a quantifier. An occurrence of a variable is bound iff that variable falls within the scope of a quantifier, and is free otherwise. The notion of a proposition in PROPLOG is expanded by the notion

of a formula in PREDLOG. A *sentence* is still defined as something which is a valid input of the system; a *proposition* is still defined as something which, if paired with a keyword, would constitute a sentence of the language; a *formula* is defined as an expression of the language which is either a proposition, or could be made into a proposition if ranged over by an appropriate quantifier.

Propositions of PREDLOG can now be defined recursively as any formula which contains no free variables and satisfies any of the following conditions:

If ***A*** is an atomic formula and contains no free variables, then ***A*** is a proposition of PREDLOG.

If ***B*** is a proposition of PREDLOG, contains no free variables, and *is not* enclosed in brackets, then NOT(***B***) is a proposition of PREDLOG.

If ***B*** is a proposition of PREDLOG, contains no free variables and *is* enclosed in brackets, then NOT***B*** is a proposition of PREDLOG.

If ***C*** and ***D*** are both propositions of PREDLOG and do not contain free variables, then:

- (***C*** AND ***D***)
- (***C*** OR ***D***)
- (***C*** IFTHEN ***D***)
- (***C*** IFF ***D***)

Are all propositions of PREDLOG.

If ***E*** is a formula of PREDLOG, ***x*** is a variable, ***E*** contains at least one occurrence of ***x***, neither 'FORALL(***x***)' nor '#FORSOME(***x***)' were used in the construction of ***E***, ***E*** contains no free variables, and *is not* enclosed in brackets then:

- FORALL(***x***)(***E***)
- FORSOME(***x***)(***E***)

Are both propositions of PREDLOG.

If ***E*** is a formula of PREDLOG, ***x*** is a variable, ***E*** contains at least one occurrence of ***x***, neither 'FORALL(***x***)' nor '#FORSOME(***x***)' were used in the construction of ***E***, ***E*** contains no free variables, and *is* enclosed in brackets then:

- FORALL(***x***)***E***
- FORSOME(***x***)***E***

Are both propositions of PREDLOG.

Nothing else is a proposition of PROPLOG.

A sentence of PREDLOG can then be defined in exactly the same way as a sentence of PROPLOG:

If ***E*** is a proposition of PREDLOG, then:

- #PREMISE ***E***
- #CONCLUDE ***E***
- #ASSUME ***E***
- #DERIVE ***E***

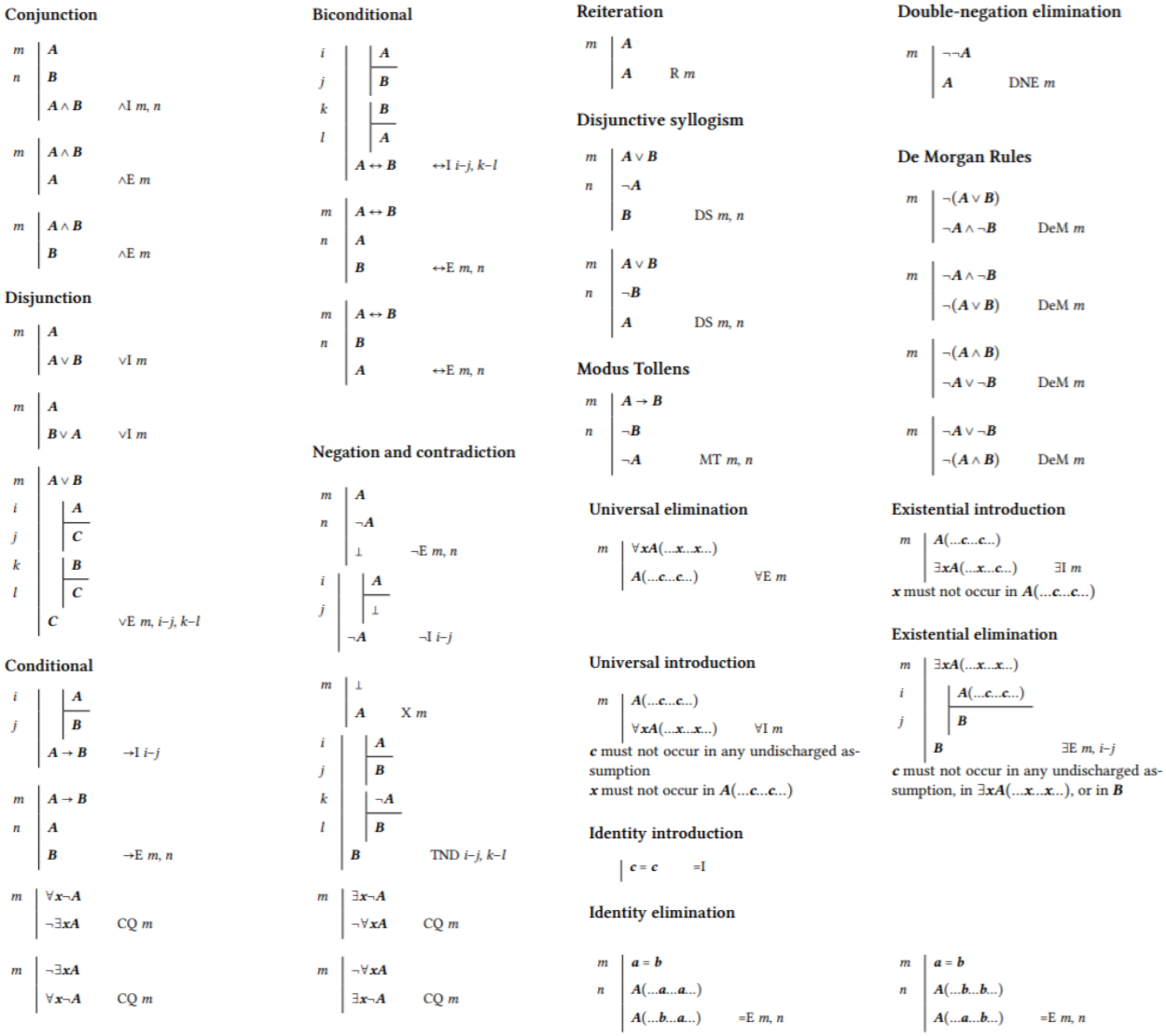
  Are all sentences of PREDLOG.

#ENDASSUME is a sentence of PREDLOG.

Nothing else is a sentence of PREDLOG.

4. Inference Rules

The following diagrams, copied from the textbook forallX:Cambridge (Magnus and Button, 2018) , represent all of the rules of formal logic implemented by the PRODLOG system:



References:

Direct source of the diagrams on page 5, and the system of formal logic which underpins the entire PRODLOG system, is:

Magnus, P. and Button, T. (2018). *forallx:Cambridge*. [online] Available at:

http://www.homepages.ucl.ac.uk/~uctytbu/forallxcam.pdf.