LogicPalet Version 10.x.x.x

This software tool is designed to help students master the basic concepts of symbolic logic, with emphasis on the semantic aspects of it. It is written by Jan Denef.

As of version 10.0 the software comes in 3 flavors: LogicPaletWin for Windows, LogicPaletMac for Mac, and LogicPaletLinux for Linux. These are desktop apps and use on each platform a native user interface (based on Wpf for Windows, Cocoa for Mac, and Gtk for Linux). The software is written in DotNet Standard 2.0, using Eto.Forms 2.4 for the user interfaces. Get it at http://www.logicpalet.com!!!

Short Overview. The software enables you to write logical formulas very fast by clicking on buttons, and to find any syntax errors. LogicPalet can verify whether or not a logical formula is true in a given **GeoWorld**: this is a structure consisting of figures on a chessboard. Such a GeoWorld can be easily constructed and graphically presented by LogicPalet. It can also explain why a formula is true (or false) in a given GeoWorld. Using the software IDP as a plugin, LogicPalet can generate a GeoWorld satisfying given conditions. Moreover, LogicPalet can determine whether or not a logical formula is true in a given **DecaWorld**, and explain why. A DecaWorld is a structure with less than ten elements and any number of unary and binary relations, and constants. It also enables you to enter the description of a DecaWorld into your computer very fast. Using the software SPASS as a plugin, LogicPalet can verify whether two formulas are logically equivalent, or whether a formula is a logical consequence of given formulas. This is the component **AskSpass.** Another component of LogicPalet is the P**roofAssistant** which enables students to build correct formal proofs using very simple deduction rules. Moreover LogicPalet supports online **OnlineHomework**: the students have to solve exercises posted on the internet by the instructor. The grading of the homework is done online and the results are sent to a database. LogicPalet also contains the **HomeworkEditor** to enable instructors compile exercises and post them on the internet. Having all these components integrated in one, makes LogicPalet a versatile didactical tool.

A very important kind of exercises is to translate statements in natural language to logical formulas. If a translation is faithful (in the sense that its correctness does not depend on the special meaning of the used relations) then it is logically equivalent to any other faithful translation. The student can use AskSpass to verify whether his translation is correct (and faithful) by comparing it with the translation of the instructor (this is done automatically for online homework). For statements about GeoWorlds the LogicPalet automatically tries to generate a counterexample if the translation is wrong. Extensive experiments at KULeuven show that the plug-ins SPASS and IDP are almost always powerful enough for these tasks.

We recommend instructors to avoid translation exercises involving equivalence relations, because the notion of faithful translation is not very natural in that case. Alternatively you can only require that the translation is faithful with respect to a specified set of constraints (e.g. the constraint that a given relation is an equivalence relation). The homework component of LogicPalet can verify whether the student's translation is faithful with respect to specified constraints.

The logical formulas have to be written in Unicode syntax.

Unicode syntax

∧	and	⇒	implication	¬	negation	∀	for all
∨	or	⇔	equivalence	≠	not equal	∃	there exists

Unicode syntax for quantifiers:

∀x: Triangle(x) ∃x: Triangle(x)

Don't forget the colon! The scope of a quantifier is all what follows after the colon, unless brackets regulate otherwise (weak quantifier binding). Blanks are irrelevant, except that they are not permitted inside variables, constants, relation and function identifiers.

Variables, constants, relation identifiers and function identifiers are strings consisting of letters or digits, starting with a letter and not containing any blanks. Such strings are called identifiers. You can use three kinds of brackets: ( ) { } [ ].

Operator binding

(strongest binding) ¬ ∧ ∨ ⇒ ⇔ ∀ ∃ (lowest binding)

For identical connectives, the binding increases from left to right. For example:

P ∧ Q ∧ R is P ∧ (Q ∧ R)	P ∧ Q ⇒ R is (P ∧ Q) ⇒ R
P ∧ Q ∨ R is (P ∧ Q) ∨ R	P ⇒ Q ⇔ R is (P ⇒ Q) ⇔ R
P ⇒ Q ⇒ R is P ⇒ (Q ⇒ R)	¬P ∨ Q is (¬P) ∨ Q

Alternative syntax. Instructors can change settings so that the LogicPalet works with other syntax preferences: for example using → and ↔ instead of ⇒ and ⇔; or using strong quantifier binding where the scope of a quantifier (∀x) or (∃x), with no colon, is the smallest subformula following it.

Smart Write and Jump. For example, if you click on the button "LeftOf" then LeftOf(*,*) is written in the focussed text field. Next, when you click on the button "x" then the first * is replaced by x and the caret moves automatically to the next *, and so on. In this way you never have to type the commas and the brackets. There are three sets of such buttons in window FormulaWriter: Default Buttons, Alternative Buttons and Custom Buttons. The Custom Buttons can be edited (i.e. changing the text that they write). Make your choice via the menu items "View > Buttons >". Smart Write and Jump is also implemented for the quantifiers, just try it!

Syntax verification. You can verify whether a formula contains no errors with respect to the Unicode syntax, by selecting one or more formulas (separated by ;) in window FormulaWriter and clicking on "Check Syntax" in the drop-down menu "Tools". This does not verify whether the names of variables, constants, relations and functions are different, or whether they have the correct number of arguments. But this can be verified by clicking on "Get Declarations" in the drop-down menu "Tools" (if there are no free variables).

Tip

Many labels, buttons and menu items have tool tips: if you point to them with the mouse, then some text will pop-up giving explanation.

AskSpass. By clicking on "A ⇔ B ???" in the drop-down menu "AskSpass", you can ask whether the formulas A and B are logically equivalent. This only works when there are no free variables, such formulas are called sentences. Your formulas have to be in the text fields A and B of window FormulaWriter. Similarly, by clicking on the menu item "AskSpass > A ⇒ B ??? " you can ask whether B is a logical consequence of A. Here A can be one or more sentences (separated by ;). Be careful to stop Spass when it is running to long and consuming to many resources. Indeed, Spass may never find an answer to your question… The plugin SPASS was developed by the Max Planck Institut Informatik. When you click on the button A, the content of the text field A is deleted and replaced by the content of the clipboard. The same holds for button B. Instead you can also drag-drop from any source into button A or B.

GeoWorld. You can make a GeoWorld by clicking on the menu item "Make a GeoWorld" in the drop-down menu "GeoWorld". This opens a new window: the GeoWorld window. A GeoWorld consists of figures on a chessboard. The figures can be triangles, squares or pentagons. The logical formulas that can be interpreted in such worlds can contain the following relation identifiers. Unary relations: Triangle, Square, Pentagon, Small, Medium, Large. Binary relations: Smaller, Larger, LeftOf, RightOf, FrontOf, BackOf. Ternary relation: Between. The interpretations of these relation symbols are the obvious ones: Larger and Smaller have to be interpreted as strictly larger and strictly smaller. LeftOf (x,y) is interpreted as "the column of x is on the left of the column of y", FrontOf(x,y) is interpreted as "the row of x is in front of the row of y", and so on. Between(x,y,z) is interpreted as "x is strictly between y and z, on the same column, same row or same (not necessarily principal) diagonal". Each figure can have at most one name. LogicPalet can determine whether or not a logical formula (of the above kind) without free variables is true in a given GeoWorld. For this you have to select that formula in window FormulaWriter, and click on the menu item "GeoWorld > Evaluate in GeoWorld ". You can evaluate more than one formula at the same time: for this you have to select all these formulas separated by ; and click on "Evaluate in GeoWorld". You can import/export a GeoWorld from/to a file or from/to the clipboard, using the drop-down menu "File" in window GeoWorld.

LogicPalet can also explain why a logical formula (of the above kind) is true (or false) in a given GeoWorld. For this you have to select that formula in window FormulaWriter, and click on the menu item "GeoWorld > Why True/False ". This opens a new window "WhyTrueFalse" providing full explanation, by clicking on the links Why? for all the components of the formula. For this it is necessary that all figures in the GeoWorld are given a name. This can be done by clicking on the menu item "Tools > Name All" in the window GeoWorld, or by editing each figure separately. Note that it is impossible to edit the GeoWorld while the window WhyTrueFalse is not closed (unless it relates to a DecaWorld).

Clicking on the menu item "GeoWorld > Generate a GeoWorld", the software tries to generate a GeoWord satisfying given conditions. See the tooltips for more information. This uses the plugin IDP that was developed by the group Knowledge Representation and Reasoning (KRR) at the KULeuven.

GeoWorld is inspired by, but different from, the commercial software called Tarski's World published by CSLI at Stanford University. The component GeoWorld of LogicPalet is not a product of CSLI.

An important kind of exercises is to determine whether or not a given logical formula is true in a given GeoWorld or DecaWorld. After finding the answer, without using LogicPalet, the student uses the software to see whether his answer is correct. If his answer is not correct, then the student can find his error using the tool "Why True/False" as explained above.

DecaWorld. You can make a DecaWorld by clicking on the menu item "Make a DecaWorld" in the drop-down menu "DecaWorld" of window FormulaWriter. This opens a new window: the DecaWorld window. A DecaWorld is a structure with less than ten elements and any number of unary and binary relations, and constants. You can choose the names of these relations as you wish (any identifier is allowed). The elements of the universe have to be the digits 1, 2, 3, …, N, with N < 10. Functions are not allowed, but you can work with as many constants as you want. In the description of the relations and constants, blanks are irrelevant. For the description of the unary relations: write the elements without commas between them. For the description of the binary relations: write the tuples (without commas between them) in condensed form (aA,bB) = (a,b)(a,B)(A,b)(A,B). For example (12,345)(5,1) is the condensed representation of the following set of tuples: (1,3)(1,4)(1,5)(2,3)(2,4)(2,5)(5,1). In this way relations can be represented in a very practical way! You don't have to memorize these conventions: if you point with the mouse to the column headers in the DecaWorld window, then the tooltips will popup. You can import/export a DecaWorld from/to a file or from/to the clipboard, using the drop-down menu "File" in window DecaWorld.

LogicPalet can determine whether or not a logical formula without free variables is true in a given DecaWorld. For this you have to select that formula in window FormulaWriter and click on the menu item "Evaluate in DecaWorld" in the drop-down menu "DecaWorld". You can evaluate more than one formula at the same time: for this you have to select all these formulas separated by ; and click on "Evaluate in DecaWorld".

LogicPalet can also explain why a logical formula (without free variables) is true (or false) in a given DecaWorld. For this you have to select that formula in window FormulaWriter, and click on the menu item “DecaWorlds> Why True/False”. This opens a new window “WhyTrueFalse” providing full explanation, by clicking on the links Why? for all the components of the formula. It is necessary that all elements in the universe of the DecaWorld are given a name. This can be done by clicking on the menu item “Tools > Name All” in the window DecaWorld. Note that it is impossible to edit the DecaWorld while the window WhyTrueFalse is not closed (unless it relates to a GeoWorld).

Online Homework. For online homework, click on the menu item “Homework > Homework Tasks” in window FormulaWriter. A new window “OnlineHomework” will open. Click there on the button “Help” and *follow the instructions*. Remember that you have to be connected to the internet. The instructor can choose between two grading policies. One policy is that only the grades of correctly solved exercises are kept on the server and that the student can keep trying. Another policy is that the student has to try to get it right at once. If the answer is wrong the first time then the grade "wrong" is kept on the server, but the student can keep trying and eventually get the grade "correct but not at first". Each exercise assignment can have a different policy. The grading policy is shown in the status bar at the bottom of the window. The date and time of when an exercise is solved by the student, is not sent to the database, to protect privacy. By clicking on the button “Grades” the student can see his grades.

An important kind of exercises is to determine whether a given formula B is a logical consequence of some given formulas A_1, A_2,... If it is a logical consequence, then the student has to give a formal proof. This can be done by the ProofAssistant. If it is not a logical consequence then the student has to give a counter example, that is a structure in which A_1, A_2,... are true and B false. Often one can take a DecaWorld for such a structure, and then the student can verify with the software whether his DecaWorld is indeed a counterexample.

ProofAssistant. The student can construct formal proofs using the ProofAssistant. Click on the menu item “Proofs > ProofAssistant” in window FormulaWriter to open the window ProofAssistant. With this tool one constructs correct formal proofs using the KE deduction rules, which are much simpler than the deduction rules used in most other existing proof assistants. Such proofs are called KE proofs. The tool is inspired by, but different from, the software WinKE developed by Ulle Endriss.

To construct a KE proof that a given sentence B is a logical consequence of a given set T of sentences, one has to derive a contradiction from T and the negation of B, using the KE rules of deduction. It is allowed to make *case distinctions*. For any sentence C one can split up the proof (at any stage of its construction) in two cases: the case that C is true and the case that C is false. This yields two *branches*. Usually one takes for C a sentence that appears already in the proof or a sub-sentence of such a sentence. Using the tool, one first selects such a sentence before clicking on the button “Case Distinction”. Repeated case distinctions yield a *proof tree* which might have many branches. A KE proof is complete if each branch contains a *contradiction*, meaning that the branch contains both a sentence and its negation. Such a branch is called a *closed branch*.

A KE deduction is always based on sentences that appear in the same branch and results in appending the conclusions to that same branch. In the ProofAssistant that branch has to be *activated* first (unless there are no case distinctions), by selecting the corresponding case and clicking on the button “Activate Branch”. To apply a deduction rule, first select the sentences on which to apply the rule (the assumptions) and click on the button for that rule. The results of the deduction are then appended to the activated branch. After you obtained both a sentence and its negation in the activated branch, you have to select these two sentences and click on the button “Contradiction” to formally close that branch. Your KE proof is complete when all branches have been closed in this way.

Sometimes, after applying a deduction rule, some of its assumptions might not be useful anymore in the currently activated branch. Such sentences *can be marked* in order to remind the user not to use it anymore in the activated branch. Attention: it might be necessary to *unmark* some of these when another branch is activated!

For simplicity of the deduction rules, *the ProofAssistant does not support using formulas which contain an inequality* $\neq$. This does not hurt because an inequality can always be replaced by the negation of an equality.

Next we give a complete list of the KE rules of deduction.

The KE Deduction Rules

The ∃ Rule. From ∃x: A(x) one deduces A(c), where c is a *new constant* that does *not* yet appear in the current branch, and A(c) is obtained from A(x) by replacing each free occurrence of the variable x by c. Moreover, from ¬∀x: A(x) one deduces ¬A(c), with c and A(c) as above.

The ∀ Rule. From ∀x: A(x) one deduces A(t), where t is any *term without variables*, and A(t) is obtained from A(x) by replacing each free occurrence of the variable x by t. Moreover, from ¬∃x: A(x) one deduces ¬A(t), with t and A(t) as above.

The Propositional Rule. The following table shows the deductions that are allowed by the Propositional Rule, where $\underline{\neg A}$ denotes the complement of A. The complement of a sentence A is C when A is of the form ¬C, and else the complement of A is ¬A.

Assumption 1	Assumption 2	Conclusion 1	Conclusion 2
A ∧ B		A	B
A ∨ B	$\underline{\neg A}$	B	
A ∨ B	$\underline{\neg B}$	A	
A ⇒ B	A	B	
A ⇒ B	$\underline{\neg B}$	¬A	
A ⇔ B	A	B	
A ⇔ B	B	A	
A ⇔ B	$\underline{\neg A}$	¬B	
A ⇔ B	$\underline{\neg B}$	¬A	
¬¬A		A	
¬(A ∧ B)	A	¬B	
¬(A ∧ B)	B	¬A	
¬(A ∨ B)		¬A	¬B
¬(A ⇒ B)		A	¬B
¬(A ⇔ B)	A	¬B	
¬(A ⇔ B)	B	¬A	
¬(A ⇔ B)	$\underline{\neg A}$	B	
¬(A ⇔ B)	$\underline{\neg B}$	A	

An instance of the Propositional Rule that uses only one assumption is called a *Unary* Propositional Rule. When two assumptions are used it is called a *Binary* Propositional Rule.

The Equality Rules. When the given sentences contain equalities or inequalities one also needs the Equality Rules. We now describe the equality rules in case that the given sentences do *not* contain inequalities:

The Assumption-free Equality Rule. Using no assumptions one deduces any equality of the form t = t, where t is any *term without variables.*

The Unary Equality Rule. From any sentence A one deduces any sentence B obtained from A by replacing (possibly several times) any sub-formula of the form t = u by u = t, where t and u are terms.

The Binary Equality Rule. From any sentence A and a sentence t1 = t2 (with t1 and t2 terms without variables), one deduces any sentence B obtained from A by replacing (possibly several times) t1 by t2 and/or t2 by t1, and/or replacing (possibly several times) any sub-formula of the form t = u by u = t, where t and u are terms.

Note that the Unary Equality Rule is a special case of the Binary Equality Rule.

When the given sentences contain inequalities then there is an additional equality rule which states that one can replace any inequality t≠u appearing in a sentence by ¬t=u. Clearly it is simpler to replace all inequalities by negations of equalities before one starts using the ProofAssistant. That is the reason why the ProofAssistant does not support inequalities.