Investigating the Shape of Tableaux in RM_3 and Ł$_3$

Bachelor Project

Ludi van Leeuwen s3092836, l.s.van.leeuwen@student.rug.nl,
Supervisor: Prof. dr. L.C. Verbrugge

Abstract. Solving logical inferences by hand is not the most efficient way of doing logic. An automated theorem prover can help generate tableau proofs and countermodels for different logics. This automation can make it possible to investigate the structure of tableaux on a larger scale. Investigating differences between structures can be done by looking at the edit distance: the number of changes that need to be made in order to transform one structure into the other. By applying edit distances to tableaux for Ł$_3$ and RM_3, as generated by an automatic tableaux solver, the causes of differences in structures of tableaux can be investigated. The differences in structure arise from the different closing rules and the different interpretations of the conditional in both logics.

1 Introduction

In this bachelor thesis, I will develop an efficient automatic tableau solver for the three-valued logics RM_3 and Ł$_3$. I will use inferences in order to investigate the different shapes that tableaux can take, and see how they differ between those two different logics, by means of looking at the edit distance.

Three-valued logics form a branch of non-classical logic, and a subset of the multi-valued logics. In classical logic, there are only two truth values: truth (1) and falsehood (0). However, these two truth values might not be adequate for realistically dealing with the world: sometimes a sentence can be both true and false, or neither true nor false. This concept can be expressed by the truth value i.

1.1 A Short Philosophical Basis of Three-Valued Logic

In 1918, the Polish logician Jan Łukasiewicz resigned as professor at Warsaw to work for the Polish ministry of education. His farewell speech was preserved in a student newspaper (Lejewski, 1973), and in it, Łukasiewicz explains his justification for three-valued logic. It was a consequence of his problems with the idea of 'necessity', or free will: if the universe is deterministic, we cannot make free choices.

Aristotle's logic, with its values of 1 and 0, adheres to this worldview: it is not possible, from a set of premises, to arrive at anything unexpected. Everything has been set in the stone of the law of necessity. Łukasiewicz did not like this: there had to be space for something else, for "possibility": "Possible events are not effects of causes, although they can be the beginning of a causal chain. An action of a creative individual can be free and yet have influence on the course of the world.", (Łukasiewicz, 1968).

This is the philosophical basis of Łukasiewicz's work: a rejection of the idea that the future is as determined as the past, the only difference being that the past has already happened (McCall, 1973). Instead, some things will be indeterminate, or i.

1.2 Three-Valued logics

The automatic tableau solver of this project solves tableaux in two different three-valued logics: Ł$_3$ and RM_3.

1.2.1 Ł$_3$

Ł$_3$ is a three-valued logic based on Łukasiewicz's reasoning as outlined above. Its three values are 1 (truth), 0 (falsehood) and i (indeterminate),

V ={0, i, 1}, D={1}. The designated value is 1, which means that an argument is only valid iff, whenever all premises have a truth value that is designated, the conclusion is also designated. Indeterminate, or i, is interpreted as being neither true nor false. This logic does not allow for situations where a sentence is related to true and also related to false (exclusion).

1.2.2 RM_3

RM_3 is a three-valued logic based on the relevance logic R. Relevance logic was a reaction to the classical meaning of the implication: In non-relevant logics, the antecedent of an implication can be unrelated to the consequent, and the whole statement can be valid: "If is a plant in my room, then aliens exist or aliens don't exist". In this case, the consequent is true, but the antecedent seems hardly relevant. This is where the variable sharing principle comes in: inferences are only valid if A and B have a propositional variable in common (Mares, 2014). R-Mingle (RM) is a quasi-relevant logic related to R: the mingle axiom $(A \to (A \to A))$ is added. Then, RM_3 is a three-valued extension of RM, with $(A \vee (A \to B))$ added. RM_3 does not have the variable sharing principle, but it is a three-valued logic (Robles, 2016).

RM_3 is a three-valued logic where V ={0, i, 1}, D={i,1}. The difference between RM_3 and Ł$_3$ is the designated set D: In RM_3, the set of designated values includes i: a valuation of i means that a sentence is both true and false, and does not allow for situations where a sentence is not related to true, and not related to false (exhaustion).

1.2.3 Truth Tables

Truth tables for the negation, conjunction and disjunction for three-valued logics (Appendix A) are straightforward: we can adapt those of classical logic, with one extra value. However the idea of the conditional in three-valued logic is not straightforward: Ł$_3$ has a different interpretation of the conditional than RM_3, as follows.

Conditional $p \supset q$ for Ł$_3$:

	q		
p	1	i	0
1	1	i	0
i	1	1	i
0	1	1	1

Conditional $p \supset q$ for RM_3:

	q		
p	1	i	0
1	1	0	0
i	1	i	0
0	1	1	1

These tables taken from Priest (2008)

1.3 Truth Value Gluts vs Truth Value Gaps

The difference between the truth tables for Ł$_3$ and RM_3 is due to how they interpret i: i as "neither true nor false" (Ł$_3$), vs i as "both true and false" (RM_3). These different approaches can be represented as N and B, respectively, N for neither, and B for both.

N is a truth value gap. A truth value gap approach assumes that a statement has no truth value, because a sentence is indeterminate, for example, due to a denotation failure or a future contingent (Priest, 2008).

B is a truth value glut. A truth value glut follows the dialetheic theory of truth: some sentences can be both true and false, such as inconsistent laws or self-reference (Priest, 2008).

The idea of truth value gluts and truth value gaps can be united into one larger theory if we assign truth values based on a relation, instead of a function. Within a function-framework, every sentence gets precisely one of (1, i, 0) as a value. However, in a relation-framework, every sentence can relate to one or more values: a sentence can relate to true, a sentence can relate to false, a sentence can relate to both true and false (B), or a sentence can relate to neither true nor false (N) (Pelletier, 2018).

1.4 FDE and its extensions

First Degree Entailment, FDE, is a four-valued logic that uses relations instead of functions in its approach: there are four truth values: 0, N, B, and 1, which are expressed as the relation ρ of a propositional parameter to a truth value from $\{0, 1\}$, as follows:

- $p\rho 1$: p relates to 1
- $p\rho 0$: p relates to 0
- $p\rho 1$ and $p\rho 0$: p relates to both 1 and 0 (B)
- no relations: p relates to neither 0 nor 1 (N)

One method to solve a logical argument is by tableau methods. These can be adapted to deal with different logics, from classical logic to FDE. The tableau methods works by converting an argument of the shape $A_1, A_2, A_3, ...A_n \vdash B$, where $A_1, ..., A_n$ are the premises of the argument, and B is the conclusion. These premises and conclusions are then converted into an initial list: for premises: $A, +$, and for conclusions $B, -$, in First Degree Entailment, or FDE.

Every sentence on the initial list can be expanded according to tableau rules, corresponding to the appropriate logic. This results in a tree-like structure with different branches called a tableau. When a contradiction is found, the branch closes. If all sentences on a certain branch have been expanded, but no contradiction has been found, the branch remains open. This means that there is a counterexample, that proves that the argument represented by the tableau is not valid. The countermodels can be read from the branch, according to the rules of the logic, and are discussed in section 1.7. (Tableaux rules for $\wedge$, $\vee$, and $\neg$ are in Appendix A).

Closure Condition. *For an FDE tableau, a branch closes if for some wff A:*
$A, +$ and $A, -$ are on the same branch.

A tableau in Ł$_3$ has the rules of FDE, but with an added condition: exclusion. Exclusion means that a parameter cannot both relate to true and relate to false, so it has to be N, not B.

Closure Condition. *For an Ł$_3$ tableau, a branch closes if for some wff A:*
1) $A, +$ and $A, -$ are on the same branch, or
2) $A, +$ and $\neg A, +$ are on the same branch.

A tableau in RM_3 has the rules of FDE, but with an added condition: exhaustion. Exhaustion means that a parameter has to relate either to true or relate to false. It has to relate to something, so it can be B, not N.

Closure Condition. *For an RM_3 tableau, a branch closes if for some wff A:*
1) $A, +$ and $A, -$ are on the same branch, or
2) $A, -$ and $\neg A, -$ are on the same branch.

1.5 Automatic Tableau Implementations

One of the originators of the semantic tableau method was E. W. Beth (1955). Before the tableau method, an inference by means of a system of natural deduction, such as Fitch (Barker-Plummer et al., 2011). In a Fitch-style system, premises are introduced, and using derivation rules, it is shown that the conclusion follows logically.

This presented difficulties, which Beth thought could be avoided by approaching entailment in a systematic way (Beth, 1955). This is the origin of the tableau method. In his original paper, Beth suggested the possibility of a logical machine, which could solve logical inferences: with a dial, and lights: a red light would burn for a closed and valid tableau, a yellow light for a finite tableau with counterexample, and a green light for an infinite tableau. Beth stated that a machine including the green light would be impossible, but it would be possible to construct a logical machine with only the red and yellow light (Beth (1955), page 29). As the logics implemented in this project are not infinite, the aim is to create such a yellow-red-light machine.

The earliest automatic tableau implementation was written in the 1950s, by Prawitz,

Prawitz and Voghera. It implemented first-order logic (Beckert & Posegga, 1995). In the 1980s, Schönfeld implemented a classical tableau implementation in Prolog (Schönfeld, 1985). Posegga and Schmitt implemented a free-variable first-order tableaux solver in Prolog (Beckert & Posegga, 1995).

More relevant to this project is the implementation of a three-valued logic system based on Kleene's strong logic in (Beckert & Posegga, 1995), in which they give a short, high-level overview of the methods used in order to efficiently solve with tableaux. One web-based implementation of multivalued logic was created by Owings (2011). Some pre-programmed arguments used by Owings (2011) were taken as to be part of the test-set (see Appendix 6.3) for our research.

1.6 Tableau Rules for Three - Valued logics

The semantic tableau rules for RM_3 and Ł$_3$ are the same as the semantic tableaux for FDE, except for the conditional, since the conditionals for RM_3 and Ł$_3$ are not equal in truth value to ¬ A ∨ B.

1.6.1 Tableau Rules for RM_3

RM_3 has closure rules for Exhaustion, and the following rules for conditional:

+ conditional

1. $A \supset B, +$
2. $A, -$ $\quad \neg B,$ $\quad A \wedge \neg A, +$
3. $B \wedge \neg B, +$

− conditional

1. $A \supset B, -$
2. $A, +$ $\quad \neg B, +$
3. $B, -$ $\quad \neg A, -$

+ negated conditional

1. $\neg(A \supset B), +$
2. $A, +$
3. $\neg B, +$

− negated conditional

1. $\neg(A \supset B), -$
2. $A, -$ $\quad \neg B, -$

1.6.2 Tableau Rules for Ł$_3$

Ł$_3$ has closure rules for Exclusion, and the following rules for the conditional:

+ conditional

1. $A \supset B, +$
2. $\neg A, +$ $\quad B, +$ $\quad A \vee \neg A, -$
3. $B \vee \neg B, -$

− conditional

1. $A \supset B, -$
2. $A, +$ $\quad \neg B, +$
3. $B, -$ $\quad \neg A, -$

+ negated conditional

1. $\neg(A \supset B), +$
2. $A, +$
3. $\neg B, +$

− negated conditional

1. $\neg(A \supset B), -$
2. $A, -$ $\quad \neg B, -$

1.7 Countermodels

A countermodel is a way of showing that an inference is not valid. If a branch is complete, but does not close, an inference is not valid. The information that is on the branch can be used to form a countermodel. In case of Ł$_3$ and RM_3, this is done by assigning relations based on the propositional parameters and the signs.

1.7.1 Ł$_3$

If there is an open and complete branch in an Ł$_3$ tableau, the countermodel can be read as follows: for every propositional parameter p, if:

- $p, +$ is on the branch, set $p\rho 1$
- $\neg p, +$ is on the branch, set $p\rho 0$.
- No other facts obtain.

These are the same rules as those for FDE.

1.7.2 RM_3

If there is an open and complete branch in an RM_3 tableau, the countermodel can be read as follows: for every propositional parameter p, if:

- $p, +$ is on the branch, set $p\rho 1$. Also set $p\rho 1$ if $p, -$ is **not** on the branch. In this case, $\neg p, -$ might be on the branch (if they're both on the branch, the branch closes), so you can also set $p\rho 1$ if you find $\neg p, -$ on the branch.
- $\neg p, +$ is on the branch, set $p\rho 0$. Also set $p\rho 0$ when $\neg p, -$ is **not** on the branch. If $p, -$ is on the branch, $\neg p, -$ will not be on the branch, so set $p\rho 0$.

The countermodel rules were taken from Priest (2008).

1.8 Research Questions

The goals of this bachelor project are to investigate the following:

- Can a correct tableau solver for Ł$_3$ and RM_3 be created?
- What are the causes of the differences in the structure of Ł$_3$ and RM_3 tableaux, as expressed by their edit distance?

2 Design

Choices made during the design of the test-set, and the creation process, are outlined in this section.

2.1 The test set

A set of 29 arguments was created, gathered from Priest (2008), Owings (2011), and old Advanced Logic exams. The inferences in the test-sets that are similar to inferences that students of logic would be able to solve by hand (Appendix 5.3).

2.2 The process

The process of creating a tableau, and the different classes that are involved, are described in the following subsections.

2.2.1 Flow through the program

The program [1] is written in Python, and consists of seven classes: `main`, `parsingExpression`, `createTab`, `countermodel`, `comparisonTableaux`, `statistics`, and `plotting`. The flow through these classes is semi-linear: arguments are taken as input in `main`, either by the user through `parsingExpression`, or preprogrammed arguments that are already in `main`. The tableau is created in `createTab`, which is called from `main`. The class `createTab` makes a tree of the arguments, creates a tableau, and calls `countermodel`, which creates the countermodel and the checked countermodel, if they exist. The tableau, countermodel and checked countermodel, are returned to `main`, where relevant information is stored in objects from the `statistics` class. The class `comparisonTableau` is used to find the edit distance between the two tableaux in Ł$_3$ and RM_3. The class `plotting` uses the statistics gathered in the `statistics` class to create plots.

2.2.2 Building the tableau in `createTab`

In this program, the input is given either as a preprogrammed argument, or by the user, who can then also indicate the logic system used for

[1] which can be found here: `https://github.com/aludi/bachelorProject`

solving. The sentences of the input, in normal form, are translated into Reverse Polish notation, using Dijkstra's shunting yard algorithm (Wolf, n.d.). From the Reverse Polish form, a sentence tree is constructed. These sentence trees are part of a TabNode instance.

Every sentence of the argument is then placed into the argument tree, or tableau (Figure 1). This argument tree is based on the tableau structure that is used to solve FDE, Ł$_3$ and RM_3, and should be familiar to the user. In the tree, the sentence is shown, as well as the sign, and the different branches. If a branch is closed according to the rules of the logic, the final node is "CLOSED".

The point of constructing a tableau is to close all branches when an argument is valid, and to provide a countermodel when an argument is invalid, which is based on an open and complete branch. The process of building the argument tree or tableau stops either when all branches are closed and the argument is valid, or when all nodes are expanded, but some branch is still open, which means that the argument is invalid. In this case, a countermodel is given.

In the process of creating a tableau, every sentence that can be expanded (sentences that are not literals, and sentences that have not already been expanded), is added to a sorted list. This list is sorted based on the branching degree of the highest priority operator. The sentences that have the lowest branching degree will be expanded first, for efficiency's sake. The branching degree of a sentence depends on its sign and the highest priority operator. I chose to implement double negation as the lowest branching degree, since this is what I usually start with when solving a tableau. The rest of the branching degrees are determined by the tableau rules for Ł$_3$ and RM_3.

Once the branching degree has been determined, new TabNodes are created. When a sentence is expanded, new nodes are added to the tableau. These new nodes, TabNodes, are based on the main operator and the sign, following the tableau rules. The sub-tree on the left side of the main operator, and the sub-tree on the right side of the main operator (Figure 2), are used to create the new TabNodes. So, for example, if $p \wedge \neg p$ is the initial sentence, the main operator is $\wedge$, the left sub-tree is p and the right sub-tree is $\neg p$. The new TabNodes are based on these two subtrees. The TabNodes contain the new sentences that are added to the tableau. They are added to all appropriate branches, and have not been expanded yet, so they are added to the sorted list of to-be-expanded nodes.

If all branches are closed, or there are no more nodes to expand, the finished tableau is printed. If the inference is valid, the program stops and moves on to the next argument. However, if it isn't valid, the countermodel is printed.

The countermodel is determined by moving up an open and complete branch from one leaf, towards the root, and looking at the literals found there. The rules for determining the relations of atoms within countermodels depend on the logic used.

```
└── p&(~p) +
    └── q|(~q) -
        └── q -
            └── ~q -
                └── p +
                    └── ~p +
                        └── CLOSED
```

Figure 1: Example of an argument tree for the argument: $(p \wedge \neg p) \vdash (p \vee \neg p)$

2.3 Sentences

One argument consists of two types of sentences: premises, and conclusions. All sentences are instances of the python class TabNode. This is inherited from the AnyNode class, which is included in the anytree package. The TabNode class has several properties: name, sign, tree, expanded, parent and children. These properties all represent useful information of the sentence.

2.3.1 Name

The name of the sentence is the well-formed formula, or wff, in Reverse Polish notation. Reverse Polish notation disambiguates the input: instead of dealing with parentheses, the highest priority operation is always found at the end of the sentence. For example, the infix wff $P \wedge \neg P$, becomes $PP\neg\wedge$. In this way, constructing the tree of the sentence that is stored in Tree can be done by moving through the Reverse Polish sentence, instead of dealing with parentheses. The Reverse Polish is not shown to the user, as the sentences are translated back to normal form when they are shown in the tableau.

2.3.2 Sign

All premises are assigned the $+$ sign, and the conclusion is assigned the $-$ sign.

These signs show the relationship between sentence and truth value:

1. $+$: Relates to true.
   (a) $A, +$: A relates to true
   (b) $\neg A, +$: Not A relates to true
2. $-$: Does not relate to true
   (a) $A, -$: A does not relate to true
   (b) $\neg A, -$: Not A does not relate to true

This was implemented separately from the Name node, since in this way, the property is easily accessible when checking closure conditions and countermodels.

2.3.3 Tree

Every sentence stores the tree of the sentence within "Tree". This is used when a sentence is expanded: the root in the tree shows which operation will be happening, and the branches on either side of the operation will become the new operators. This was implemented in order to find quickly what the highest priority operator is, and its span.

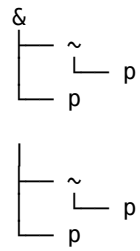

Figure 2: Examples of two sentence trees: one for $(p \wedge \neg p)$, and one for $(p \vee \neg p)$.

2.3.4 Expanded

The Expanded attribute shows if a sentence has already been expanded. It is not visible from a sentence whether it has already been expanded, unless it is a literal. Since every occurrence of a sentence is only expanded once, the Expanded attribute is useful in figuring which sentences still need to be expanded, and which branches are open and complete.

2.3.5 Parent and children

The sentences are part of the argument tree, which means that they will have a parent and children as well. These children and the parent are also sentences, instantiated in TabNode. All sentences from the input are initially added on the same branch, all coming from root. Then, when the sentences are expanded, new branches will be created.

2.4 Quick Countermodel

A feature for quick countermodels was implemented in the code. This feature means that the first open branch where all nodes are expanded, is taken and read as a countermodel. For short, or closed tableaux, this should result in slightly longer runtimes than the normal countermodel strategy, since it is implemented in an extra checking step. However, for longer tableaux, or a situation where you want to find

the first countermodel instead of generating the whole tableau, this feature will be advantageous. It should result in shorter run-times for large, invalid arguments, since a countermodel will usually be found before all sentences are expanded.

2.5 Infix Notation

Even though all the computations are done in Reverse Polish notation to avoid dealing with ordering ambiguity, the final representation of the sentences in the tableau is in infix notation. This infix notation is acquired by traversing the sentence-tree in-order, through recursion. In this way, the user will see notation that she is familiar with, which will improve usability.

2.6 Double-checking Countermodels

All countermodels are automatically checked. This is done by assigning to a literal, the truth values that were found in the countermodel, then evaluating every part of the sentence recursively. A countermodel is generated when a tableau doesn't close, and an argument isn't valid: all premises have a truth value that is designated, but the conclusion does not have a truth value that is designated. In the check for the countermodel, I'm assigning each atom the truth value found in the countermodel, as follows:

For RM_3, for some propositional parameter p, if:

- `p relates to 1`, assign 1 to p,
- `p relates to 0`, assign 0 to p,
- `p relates to 1 and p relates to 0`, assign i to p.

For Ł$_3$, for some propositional parameter p, if:

- `p relates to 1`, assign 1 to p,
- `p relates to 0`, assign 0 to p,
- `no facts obtain about p`, assign n to p. This is a placeholder for a non-relation, and is used during higher-level recursion rules in determining the truth-values of the premises.

In the program, every premise and conclusion is printed in infix-form, with its truth value next to it. In this way, the user can see that, in an invalid argument, all premises are designated, but the conclusion isn't.

2.7 Edit distance

It is useful to compare two tableaux to see how they differ in form. The two tableaux of an inference in Ł$_3$ and RM_3 were converted to a string, and the Levenshtein distance was implemented (Navarro, 2001). The Levenshtein distance is calculated by looking at the number of 'edits' that need to be made to generate two identical strings. Each edit operation has a cost of 1: one can delete, swap, or insert characters. A distance of 0 means that the tableaux are identical, while a larger distance means that the tableaux are very different.

3 Results

Out of the 29 inferences in the test set, 17 inferences were valid in Ł$_3$, and 16 inferences were valid in RM_3.

The number of nodes associated with the runtime for both logics is shown in Figure 3. The number of rules applied to the complete tableaux are shown in Figure 4. The edit-distance between two complete tableaux per inference is shown in Figure 5. For graphs on the runtime, and on the number of branches generated in each logic, see Appendix 6.1.

To see whether the differences in runtime and number of rules applied were significantly different, a pairwise t-test was performed. This t-test compared the values for the same inferences in Ł$_3$ and RM_3. M is the mean of each data set,

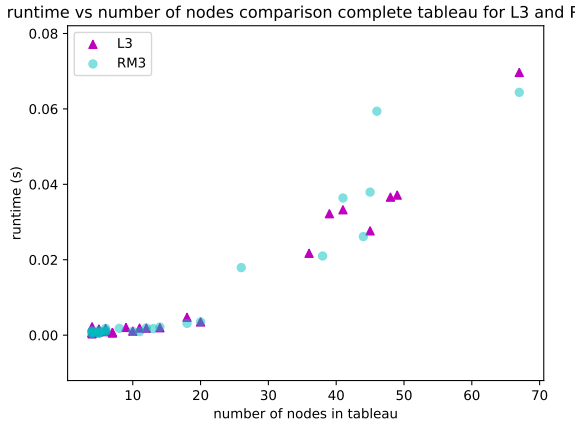

Figure 3: Comparing the number of nodes and the runtime for the same arguments in Ł$_3$ and RM_3

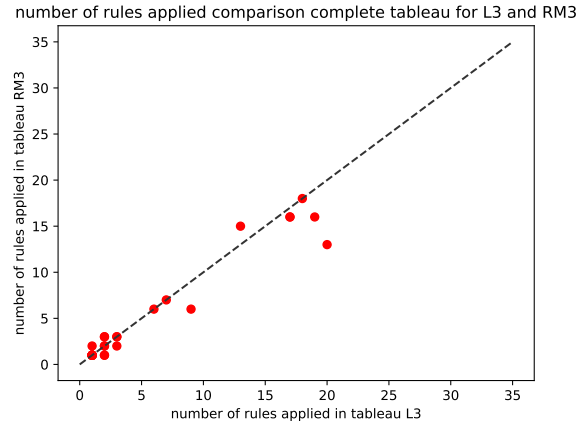

Figure 4: Comparing the number of rules applied in a complete tableau for solving the same arguments in Ł$_3$ and RM_3

and sd is the standard deviation of the data set. A p-value of 0.05 was used.

3.1 Runtime

In Figure 3, there is an exponential increase in runtime for both Ł$_3$ and RM_3: For tableaux with a number of nodes in range 0, 20, the differences in runtime are fairly small, all falling between 0.0s and 0.01s. As the number of nodes in the tableau increases, the runtime increases in a parabola-like fashion. However, a pairwise t-test of the difference in runtime does not produce significant results: Ł$_3$ ($M = 0.01, sd = 0.01$), RM_3 ($M = 0.01, sd = 0.02$), ($T(29) = 0.12, p > 0.05$).

3.2 Number of Rules Applied

In Figure 4, the number of rules applied in RM_3 ($M = 5.0, sd = 5.71$) is mostly equal to the number of rules applied in Ł$_3$ ($M = 5.44, sd = 6.43$). There is no significant difference between the number of rules applied in Ł$_3$ tableaux vs rules applied in RM_3 tableaux: ($T(29) = 1.49, p > 0.05$).

3.3 Edit Distance

In many cases, the tableaux for the same inference in for two different logics with different closing rules are identical. Out of the 29 tested inferences, 12 tableaux were identical, with an edit distance of 0.

This leaves 17 non-identical tableaux. The causes for their dissimilarity are investigated below. They can be split into three categories: differences due to validity, difference due to rule expansion, and differences due to the interpretation of the conditional.

1. For 4 inferences, the edit distance between the two tableaux was 1 (marked with '*' in the table in the appendix). The number of rules applied in these tableaux was the same, but in one logic, the inference was valid, and the tableau closed, and in the other, it was not valid, and the tableau did not close (Figure 6).

2. There are 2 inferences (marked with '$' in the table in the appendix), with an edit distance of 5: $p \wedge \neg p \vdash p \vee \neg p$ and $p \wedge \neg p \vdash q \vee \neg q$. They are valid in both logics. The differ-

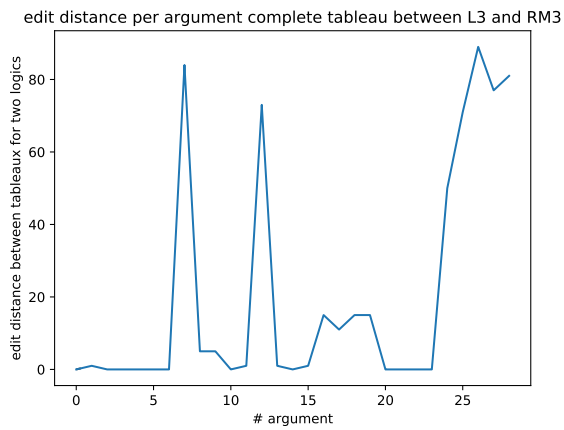

Figure 5: Comparing the edit distance for the same arguments in Ł$_3$ and RM_3

ences in edit distance are due to the fact that Ł$_3$ has to expand one extra node in order to close the tableaux. The difference here is not due to the fact that they are fully expanded, and valid in one logic, but not in the other one. Instead, these differences arise from the fact that one logic (in this case RM_3), can close branches in fewer steps (Figure 7).

3. For 11 other inferences (marked with '%' in the table in the appendix), the edit distance was between 11 and 89. This is partially, but not entirely related to the number of conditionals in the inferences (Figure 8). This is not surprising, as the interpretation of the conditional rule is one of the points where the two logics diverge. It is unclear at this point what causes the variation between the edit distances.

4 Conclusion and Discussion

In this section, a conclusion is drawn about the effectiveness of the solver, problems with the solver are discussed, and ideas for further research are explained.

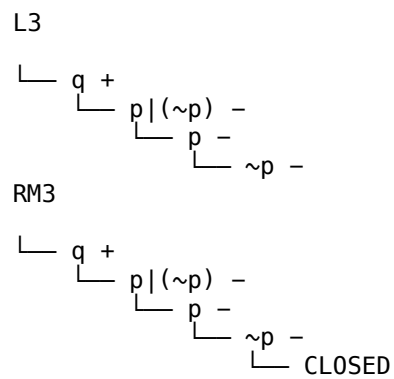

Figure 6: Example of an edit distance of 1 for $q \vdash_{RM_3} p \vee \neg p$

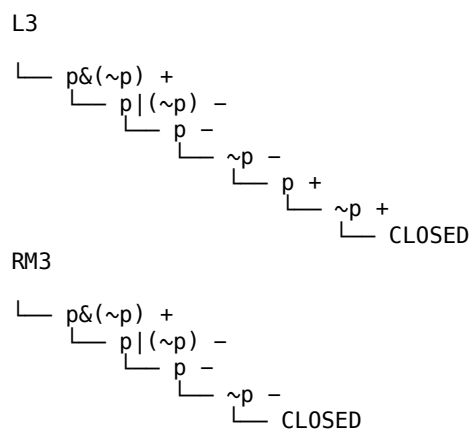

Figure 7: Example of an edit distance of 5 for $p \wedge \neg p \vdash p \vee \neg p$

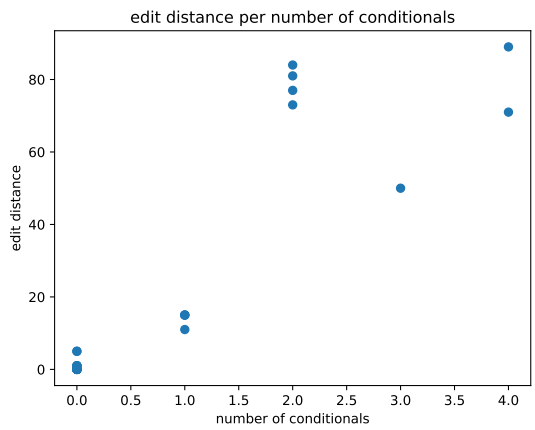

Figure 8: The edit distance between two tableaux over the number of conditionals in the inference. There is a rising trend, but this does not explain all variation.

4.1 Conclusion

The inferences in the test set were solved correctly for both logics, Ł$_3$ RM_3. The solver is as fast at solving inferences in Ł$_3$ as at solving those same inferences in RM_3, so there is no significant difference in runtime between the two logics. There is also no significant difference between the number of rules applied for the same inferences in both logics. There were 12 identical tableaux, and 17 non-identical tableaux between the two logics.

The reasons for the non-identical tableaux can be classified in three classes: 1) differences due to validity, where two tableaux are almost identical, only the inference is valid in one logic, but not in the other, 2) differences due to rule expansion, where the two inferences are both valid, but one logic has to apply more rules in order to close the tableaux. Class 1 and class 2 are both caused by the differences in closure rules between the logics.

The third class are the differences due to the interpretation of the conditional. The edit distance for these cases was the largest. A general trend can be observed: increasing the number of conditionals means an increase in edit distances between tableaux. However, there is variation that is not explained by only the number of conditionals.

4.2 Discussion

The variation in edit distance is not only due to the number of conditionals in an inference. As shown in Figure 8, an inference with more conditionals does often have a higher edit distance between two logics, than an inference with fewer conditionals. However, the variation between the tableaux might be due to the different closure conditions for Ł$_3$ and RM_3 in combination with the difference in interpretation of the conditional: If at one point the tableau does not close in one logic, but does close in the other, and in the logic where the tableau doesn't close, there is a conditional that has to be expanded, the edit distance will be larger.

The selection of inferences has been limited so far: only small inferences, similar to those studied in basic logic courses, have been checked. Larger inferences, designed to 'push' the algorithm, have not been implemented. This means that the information about the runtime and number of branches of the tableaux have only been checked for smaller inferences. Given the sharp rise in runtime for inferences with four conditionals, and the disparities in runtime between RM_3 and Ł$_3$, the inclusion of a larger test-set would be useful. This test-set should have more conditionals, and more premises. This would give a better indication of the performance of this algorithm.

The tableaux are generated by always taking the least-branching option. This is similar to how humans perform, but not ideal. There are two different problems that this heuristic is not equipped to solve, illustrated by two cases:

1. $p \vee q, \neg q, r \wedge s \vdash p$. This inference is valid. The rational approach would be to exclude $r \wedge s$ from expansion, as the literals are confined to the sentence, and do not occur in the conclusion, and are not negated

at some other place in the premises. An added heuristic would be to implement a check that a sentence that consists of single-occurring literals would not be expanded.

2. $p \wedge p \wedge p \wedge p \wedge ... \wedge p \wedge q \vdash q$. This inference is valid. The rational approach would be to only expand q from $p \wedge p \wedge p \wedge p \wedge ... \wedge p$, in order to close. However, there is an explicit order of operations in the sentence trees, based on the Reverse Polish reading of the input. In this case, it would be better to not have this explicit order of operations, and instead expand at any place in the sentence. This can be solved by a user adding parentheses, and is difficult to fix with a heuristic.

4.3 Further research

At this point, only limited versions of RM_3 and $Ł_3$ have been implemented: there is no quantification, and no modality. These were not implemented, because these features can generate infinite tableaux. However, since the full logic does include these features, further research can be done in investigating how quantification and modality would change the runtime, number of branches and countermodels of $Ł_3$ and RM_3.

Based on the results found in this report, investigating any of the three causes of a non-zero edit distance would be worthwhile. Especially in the case of the conditional, it is, as of yet, unknown what causes the variation between edit distances for arguments with the same number of conditionals.

Classical logic and FDE can be seen as versions of $Ł_3$ and RM_3 with different constraint rules: FDE, a four-valued logic, has none of the extra constraint rules that $Ł_3$ and RM_3. Classical logic, a two-valued logic, has all constraints of $Ł_3$ and of RM_3. By implementing FDE and classical logic as well (where classical logic is FDE logic with both $Ł_3$, exclusion, and RM_3, exhaustion, constraint rules added), a deeper investigation into the differences between the number of branches, edit distances and runtimes for these four logics can be conducted.

References

Barker-Plummer, D., Barwise, J., & Etchemendy, J. (2011). *Language, proof, and logic: Second edition* (2nd ed.). Center for the Study of Language and Information/SRI.

Beckert, B., & Posegga, J. (1995). leantap: Lean tableau-based deduction. *Journal of Automated Reasoning*, *15*(3), 339–358. doi: https://doi.org/10.1007/BF00881804

Beth, E. W. (1955). Semantic entailment and formal derivability. *Mededelingen der Koninklijke Nederlandse Akademie van Wetenschappen Afd Letterkunde*, *18(3)*, 309-342.

Lejewski, C. (1973). Review. *The Journal of Philosophy*, *70*(3), 81-85. doi: https://doi.org/10.2307/2024840

Łukasiewicz, J. (1968). Farewell lecture at the University of Warsaw, March 7, 1918. *The Polish Review*, *13*(3), 45-47. Retrieved from `www.jstor.org/stable/25776790`

Mares, E. (2014). *Relevance logic.* Stanford Encyclopedia of Philosophy.

McCall, S. (1973). Reviewed work: Selected works by Jan Łukasiewicz. *Synthese*, *26*(1), 165-171. doi: https://doi.org/10.1007/BF00869760

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, *33*(1), 31-88. doi: https://doi.org/10.1145/375360.375365

Owings, D. (2011). *pytableaux.* Retrieved from `http://logic.dougowings.net/doc/introduction.html`

Pelletier, F. J. (2018). Peculiarities of some three- and four-valued second order logics. *Logica Universalis*, *12*(3), 493-509. doi: https://doi.org/10.1007/s11787-018-0214-7

Priest, G. (2008). *An introduction to non-classical logic: From if to is* (2nd

ed.). Cambridge University Press. doi: https://doi.org/10.1017/CBO9780511801174

Robles, G. (2016). The quasi-relevant 3-valued logic RM_3 and some of its sublogics lacking the variable-sharing property. *Reports on Mathematical Logic*, 105-131. doi: DOI: 10.4467/20842589RM.16.008.5285

Schönfeld, W. (1985). Prolog extensions based on tableau calculus. In J. Aravind (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Vol. 2, p. 730-732). Morgan Kaufmann Publishers Inc. Retrieved from `https://www.ijcai.org/Proceedings/85-2/Papers/063.pdf`

Wolf, C. E. (n.d.). *The shunting yard algorithm.* `http://www.oxfordmathcenter.com/drupal7/node/628`. (adapted by P. Oser)

5 Appendix A: Rules

5.1 Truth Tables for Three-Valued logics

Negation:

p	$\neg$
1	0
i	i
0	1

Conjunction:

	q		
p	1	i	0
1	1	i	0
i	i	i	0
0	0	0	0

Disjunction:

	q		
p	1	i	0
1	1	1	1
i	1	i	i
0	1	i	0

5.2 Tableau Rules for Three-Valued Logics

5.2.1 conjunctions

+ conjunctions

1. $A \wedge B, +$
2. $A, +$
3. $B, +$

− conjunctions

1. $A \wedge B, -$
2. $A, -$ $\quad$ $B, -$

5.2.2 disjunctions

+ disjunctions

1. $A \vee B, +$
2. $A, +$ $\quad$ $B, +$

− disjunctions

1. $A \vee B, -$
2. $A, -$
3. $B, -$

5.2.3 negated conjunctions

+ negated conjunction

1. $\neg(A \wedge B,)+$
2. $\neg A \vee \neg B, +$

− negated conjunction

1. $\neg(A \wedge B), -$
2. $\neg A \vee \neg B, -$

5.2.4 negated disjunction

+ negated disjunctions

1. $\neg(A \vee B), +$
2. $\neg A \wedge \neg B, +$

6 Appendix B: Test set and results

6.1 Extra graphs

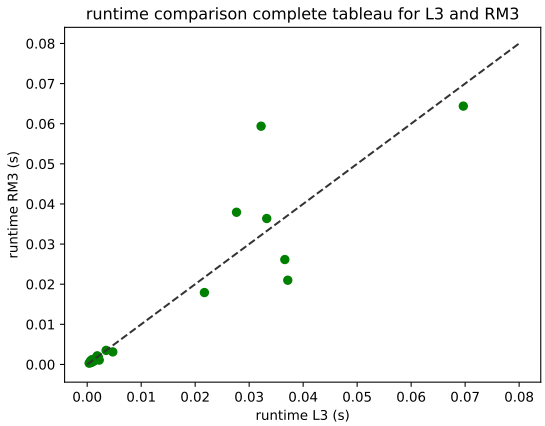

Figure 9: The test-set is ran for Ł$_3$ and RM_3. The runtimes for each inference are measured, one runtime in Ł$_3$, and the other one in RM_3. For each inference, the x-coordinate is the runtime in Ł$_3$, and the y-coordinate is the runtime in RM_3. For short inferences, the runtimes of the two logics are very similar, so they lie on $x = y$, represented by the dotted line. As the inferences are longer, runtime diverges from the middle line. There are more inferences that take longer in Ł$_3$ than in RM_3, than the other way around.

6.2 Sources of the Test Set

The inferences 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 were taken from Priest (2008), the inferences 27, 28, 29 were from old Advanced Logic exams, and the rest are from Owings (2011),

6.3 Test set of arguments

logic	arguments	valid	runtime	countermodel	countermodel check

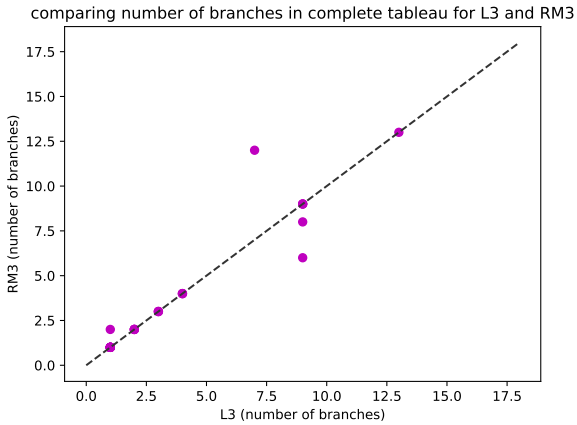

Figure 10: Comparing the number of branches in a complete tableau for the same inference in Ł$_3$ and RM_3. In some cases, the number of branches are the same in RM_3 as in Ł$_3$. However, in two cases, the number of branches needed to complete a tableaux in Ł$_3$ is higher than the number of branches needed for the same tableaux in RM_3. The number of branches in Ł$_3$ ($M = 3.66, sd = 3.72$)) was not significantly higher than the number of branches in RM_3 ($M = 3.42, sd = 3.52$) ($T(29) = -0.32, p > 0.05$.

L3	1. p&(q|~q), |- r,	False	0.00172782	q is related to 1 p is related to 1 no other facts obtain	p&(q|(~q) 1 r n
RM3	1. p&(q|~q), |- r,	False	0.00179696	q is related to 1 r is related to 0 p is related to 1 no other facts obtain	p&(q|(~q) 1 r 0
L3	2. * p&~p, |- q,	True	0.00161314	No countermodel	
RM3	2. * p&~p, |- q,	False	0.00113606	p is related to 0 p is related to 1 q is related to 0 no other facts obtain	p&(~p i q 0
L3	3. p&q, |- p,	True	0.00088501	No countermodel	
RM3	3. p&q, |- p,	True	0.00118709	No countermodel	
L3	4. p, |- p|q,	True	0.00100589	No countermodel	
RM3	4. p, |- p|q,	True	0.000641108	No countermodel	
L3	5. p&(q|r), |- (p&q)|(p&r),	True	0.00474119	No countermodel	
RM3	5. p&(q|r), |- (p&q)|(p&r),	True	0.00312495	No countermodel	
L3	6. p|(q&r), |- (p|q)&(p|r),	True	0.00350595	No countermodel	
RM3	6. p|(q&r), |- (p|q)&(p|r),	True	0.00350809	No countermodel	
L3	7. ~~p, |- p,	True	0.000357151	No countermodel	
RM3	7. ~~p, |- p,	True	0.000319958	No countermodel	
L3	8. % (p&q)>r, |- (p&~r)>q,	False	0.0371509	q is related to 0 p is related to 1 r is related to 0 no other facts obtain	(p&q)>r 1 (p&(~r))>q 0
RM3	8. % (p&q)>r, |- (p&~r)>q,	False	0.0209911	q is related to 0 p is related to 1 r is related to 0 r is related to 1 no other facts obtain	(p&q)>r 1 (p&(~r))>q 0
L3	9. $ (p&~p), |- (p|~p),	True	0.000671864	No countermodel	
RM3	9. $ (p&~p), |- (p|~p),	True	0.000520945	No countermodel	
L3	10. $ (p&~p), |- (q|~q),	True	0.000614882	No countermodel	
RM3	10. $ (p&~p), |- (q|~q),	True	0.000801802	No countermodel	
L3	11. (p|q), |- (p&q),	False	0.00107217	q is related to 1 no other facts obtain	p|q 1 p&q n

RM3	11. (p|q), |- (p&q),	False	0.000983	q is related to 1 p is related to 0 no other facts obtain	p|q 1 p&q 0
L3	12. * p, ~(p&~q), |- (q),	True	0.00205112	No countermodel	
RM3	12. * p, ~(p&~q), |- (q),	False	0.00181198	p is related to 0 p is related to 1 q is related to 0 no other facts obtain	p i ~(p&(~q) i q 0
L3	13. % (p&q)>r, |- p>(~r|q),	False	0.0332532	q is related to 0 p is related to 1 no other facts obtain	(p&q)>r 1 p>((~r)|q) 0
RM3	13. % (p&q)>r, |- p>(~r|q),	False	0.0363741	q is related to 0 p is related to 1 r is related to 1 no other facts obtain	(p&q)>r 1 p>((~r)|q) 0
L3	14. * p|q, ~q, |- p,	True	0.000677824	No countermodel	
RM3	14. * p|q, ~q, |- p,	False	0.000921965	q is related to 1 q is related to 0 p is related to 0 no other facts obtain	p|q i ~q i p 0
L3	15. p|q, p, |- q,	False	0.000951052	p is related to 1 p is related to 1 no other facts obtain	p|q 1 p 1 q n
RM3	15. p|q, p, |- q,	False	0.0010643	p is related to 1 q is related to 0 no other facts obtain	p|q 1 p 1 q 0
L3	16. * q, |- p|~p,	False	0.000682116	q is related to 1 no other facts obtain	q 1 p|(~p) n
RM3	16. * q, |- p|~p,	True	0.000462055	No countermodel	
L3	17. % p>q, ~p, |- q,	False	0.00188112	p is related to 0 p is related to 0 no other facts obtain	p>q 1 ~p 1 q n
RM3	17. % p>q, ~p, |- q,	False	0.00172305	p is related to 0 q is related to 0 no other facts obtain	p>q 1 ~p 1 q 0
L3	18. % p&~p, q, |- p>p,	True	0.00123692	No countermodel	
RM3	18. % p&~p, q, |- p>p,	True	0.000929117	No countermodel	
L3	19. % p>q, p, |- q,	True	0.00186396	No countermodel	
RM3	19. % p>q, p, |- q,	True	0.00214887	No countermodel	
L3	20. % p>q, ~q, |- ~p,	True	0.00197196	No countermodel	

RM3	20. % p>q, ~q, |- ~p,	True	0.001899	No countermodel	
L3	21. ~(p|q), |- ~p&~q,	True	0.000693083	No countermodel	
RM3	21. ~(p|q), |- ~p&~q,	True	0.000645876	No countermodel	
L3	22. ~(p&q), |- ~p|~q,	True	0.00121903	No countermodel	
RM3	22. ~(p&q), |- ~p|~q,	True	0.00125408	No countermodel	
L3	23. ~p&~q, |- ~(p|q),	True	0.00223184	No countermodel	
RM3	23. ~p&~q, |- ~(p|q),	True	0.00108695	No countermodel	
L3	24. ~p|~q, |- ~(p&q),	True	0.000701904	No countermodel	
RM3	24. ~p|~q, |- ~(p&q),	True	0.000638008	No countermodel	
L3	25. % p>(p>q), |- p>q,	False	0.0217099	q is related to 0 no other facts obtain	p>(p>q 1 p>q n
RM3	25. % p>(p>q), |- p>q,	True	0.01792	No countermodel	
L3	26. % (p>(p>q))>p, |- p>q,	False	0.032207	q is related to 0 p is related to 1 no other facts obtain	(p>(p>q))>p 1 p>q 0
RM3	26. % (p>(p>q))>p, |- p>q,	False	0.059381	q is related to 0 p is related to 1 no other facts obtain	(p>(p>q))>p 1 p>q 0
L3	27. % (p>q)&(r>s), |- (p>s)|(r>q),	True	0.0696619	No countermodel	
RM3	27. % (p>q)&(r>s), |- (p>s)|(r>q),	True	0.0643919	No countermodel	
L3	28. % (p&q)>r, |- p>(~q|r),	False	0.0276563	p is related to 1 no other facts obtain	(p&q)>r 1 p>((~q)|r) n
RM3	28. % (p&q)>r, |- p>(~q|r),	False	0.0379322	q is related to 0 q is related to 1 p is related to 1 r is related to 0 r is related to 1 no other facts obtain	(p&q)>r i p>((~q)|r) 0
L3	29. % (p&q)>r, |- (p&~r)>~q,	False	0.0366058	q is related to 1 no other facts obtain	(p&q)>r 1 (p&(~r))>(~q) n
RM3	29. % (p&q)>r, |- (p&~r)>~q,	False	0.0261462	p is related to 0 p is related to 1 q is related to 1 r is related to 0 r is related to 1 no other facts obtain	(p&q)>r i (p&(~r))>(~q) 0

The total number of inferences: 29
Inferences valid in RM3: 17
Inferences valid in L3: 18

− negated disjunctions `all prewritten arguments are resolved`

1. $A \vee B, -$
2. $\neg A \wedge B, -$

5.2.5 double negation

+ double negations

1. $\neg\neg A, +$
2. $A, +$

− double negations

1. $\neg\neg A, -$
2. $A, -$