university of groningen / faculty of science and engineering

# A SOUND AND COMPLETE NON-BRANCHING TEMPORAL LINEAR-TIME LOGIC SOLVER WITH THE EXTENDABILITY CONSTRAINT

Bachelor's Project Thesis

Daniëlle Metz, s2493403, d.m.metz@student.rug.nl,
Supervisor: Prof Dr L.C. Verbrugge

**Abstract:** In this research, a non-branching temporal linear-time logic solver was implemented with the extendability constraint. The logic was tested on its soundness and completeness. The solver was tested on how fast it could solve 6 temporal linear-time logic problems with or without allowing infinite branching. Half of the problems required extendability to get further down the tableau. The results show that these problems require significantly more time to solve the problem a thousand times with the allowance of infinite branches compared to the standard solver. There was also one problem that did not require extendability that still showed significant results. This could have been caused by an extra if-evaluation or the great variation in run-time in general.

## 1 Introduction

In this research, a temporal (or tense) linear-time logic solver will be made using the tableau method in order to solve the validity problem for both general temporal logic and temporal logic with the constraint of extendability. The solver will also be compared to an existing temporal linear-time logic solver and tested on its soundness and completeness. In both comparisons, the run-time of the systems will be compared. The research question of this paper consists of two parts, which are:

*1. Can we make a non-branching temporal linear-time logic solver that allows infinite open branches and that is sound and complete?*

*2. How fast does a non-branching temporal linear-time logic solver that allows infinite open branches solve a set of temporal linear-time logic problems compared to a normal non-branching temporal linear-time logic solver using the tableau method?*

To answer these questions, we first need to understand the terms used. Temporal logic or tense logic $K^t$ is another interpretation of modal logic $K$. As the semantics of tense logic is exactly the same as modal logic, only with more expressive language. We will explore modal logic first.

### 1.1 Modal and Temporal logic

Modal logic consists of modes where different truths are possible, namely possibility, necessity and impossibility. One possible world might hold a notion about some other possible world it is connected to. This brings us to the symbols $\Box$ and $\Diamond$, where $\Box$ is read as 'It is necessarily the case that A' and $\Diamond$ as 'It is possibly the case that A'. Using these new operators on top of the general truth functions ($\neg$, $\wedge$ and $\vee$), makes the language of modal logic $K$. An interpretation of the modal logic language is a triple $< W, R, v >$, where $W$ is a non-empty set of possible worlds, $R$ is a binary relation on $W$, and $v$ is a truth function that assigns a truth value to each atomic proposition at each possible world (Priest, 2008). We will use this triple $< W, R, v >$ in this research.

As mentioned before, tense logic $K^t$ is another interpretation of modal logic with the same semantics. The difference is the interpretation of $w1Rw2$. Whereas in modal logic $w1Rw2$ refers to an accessibility of $w2$ from world $w1$, in tense logic it means: '$w1$ is earlier than $w2$'. $\Box$ and $\Diamond$ will be replaced by $G$ meaning 'at all later times', $F$ 'at some later time', $H$ meaning 'at all earlier times' and $P$ 'at some earlier time' (Priest, 2008).

A linear-time logic means basically that the temporal

semantics follows a linear fashion. One time in relation to another is either before it, at the same time, or after it. That is, $\forall xy(xry \vee x = y \vee yrx)$.

## 1.2 The tableau method

One of the most common ways to show that an inference is valid, is by constructing a tableau using the tableau method by (Beth, 1955). A tableau is a tree structure. The tree consists of a root and nodes connected by branches; the nodes at the bottom are leaves. To test the validity of an inference, a tableau is constructed with the premises of an inference and the negation of the conclusion at the root labeled with 0, to refer to a world $w_0$. Then rules are applied which allows the tree to extend its branches. Once a literal and its negation appear in a branch in the same world, the corresponding inference is unsatisfiable and the branch is closed. When all the branches of the tableau are closed, the inference is proven to be valid. If one or more branches are open and complete, meaning every rule that can be applied has been applied, these branches serve as a counter-model for the inference (Priest, 2008). The tableau rules for $G$, and $H$ are as follows:

$$\begin{array}{cc} GA,i & HA,i \\ irj & jri \\ \downarrow & \downarrow \\ A,j & A,j \end{array}$$

To be applied for every $irj$ already appearing on the branch. The tableau rules for $F$ and $P$ are as follows:

$$\begin{array}{cc} FA,i & PA,i \\ \downarrow & \downarrow \\ irj & jri \\ A,j & A,j \end{array}$$

To be applied for a new $j$ not yet appearing on the branch.

The constraint of extendability $\eta$ is the constraint of infinity towards the future. The definition of this constraint is that if there is an $i$ on the branch, a new $j$ can be added such that $irj$.

$$\begin{array}{c} \bullet \\ \downarrow \\ irj \end{array}$$

Here, $i$ is not new but $j$ is new. The definition of $\eta'$ is that if there is an $i$ on the branch, a new $k$ can be added such that $kri$, that is, infinity towards the past:

$$\begin{array}{c} \bullet \\ \downarrow \\ kri \end{array}$$

Here, $i$ is not new but $k$ is new.

This rule must be applied with care, as it can cause an infinite branch when called immediately after itself (Priest, 2008).

In this research, we are dealing with linear-time logic, meaning a few more tableau-rules need to be mentioned. The first ones are $\varphi$ and $\beta$. $\varphi$ correspond to not branching towards the future, that is, for all $w_1$, $w_2$, $w_3$, if $w_1Rw_2$ and $w_1Rw_3$, then $w_2Rw_3$ or $w_3Rw_2$ or $w_2 = w_3$. The tableau rule for $\varphi$ is:

$$\begin{array}{ccc} & irj & \\ & irk & \\ \swarrow & \downarrow & \searrow \\ jrk & j=k & krj \end{array}$$

$\beta$ corresponds to not branching towards the past, that is , for all $w_1$, $w_2$, $w_3$, if $w_2Rw_1$ and $w_3Rw_1$, then $w_2Rw_3$ or $w_3Rw_2$ or $w_2 = w_3$. The tableau rule for $\beta$ is:

$$\begin{array}{ccc} & jri & \\ & kri & \\ \swarrow & \downarrow & \searrow \\ jrk & j=k & krj \end{array}$$

Another rule that comes with linear time is the rule of transitivity $\tau$, which states that for all $w_1$, $w_2$, $w_3$, if $w_1Rw_2$ and $w_2Rw_3$, then $w_1Rw_3$. Its tableau rule is:

$$\begin{array}{c} irj \\ jrk \\ \downarrow \\ irk \end{array}$$

Many researchers have previously looked at automatic logic solving. In their research, Felty and Théry (1997) presented a theorem prover that implements an inference system for temporal logic. Clarke, Emerson, and Sistla (1986) have implemented a finite-state concurrent system that meets a specification expressed in a temporal logic. Kloetzer and Belta (2008) found, given a linear system and a linear temporal logic inference over a set of linear predicates in its state variables, a feedback control law with polyhedral bounds and a set

of initial states so that all trajectories of the closed loop system satisfy the inference. This is a more complicated system than will be implemented in this paper, in which we will deal with a simpler system of propositions.

For the construction of a valid logic solver, the rules will need to be sound and complete with respect to the relevant semantics. Soundness is a property of a logic system if and only if its inference rules only prove inferences that are valid according to its semantics. A formal system is complete with regard to a semantics if all inferences tha are valid according to the semantics, can be derived using the system. Soundness and, particularly, completeness proofs for logics are very simple using the tableaux method.

# 2 Methods

## 2.1 The solver

To build a temporal logic solver that is completely user-friendly and efficient, we need to consider a few things before building the solver. The first thing is that we want the solver to approach the problem in a human-like way. This means that the input should be given in a human-readable fashion and the output should look like a tableau a human could have constructed. Also, the tableau cannot skip steps in its solving. Although skipping steps would make the runtime faster, this is not the goal. Lastly, a human would give priority to certain actions above other, in order to make the tableau easier to solve for them (it might not be the fastest way to solve it). The priority of operations the solver follows is given in Table 2.1.

**Table 2.1: Priority of operations**

| |
|---|
| $\neg\vee$ and $\wedge$ |
| $\neg\rightarrow$ |
| $\vee$ and $\neg\wedge$ |
| $\rightarrow$ |
| $\leftrightarrow$ and $\neg\leftrightarrow$ |
| $\neg G$, $\neg H$, $F$ and $P$ |
| Atoms |
| $G$, $H$, $\neg F$ and $\neg P$ |

Keeping the human-like approach in mind, the solver also has to be as efficient as possible. To achieve this, the solver will be implemented in a way that it will get rid of double negation instantly. Whenever it has to add a negation on an already negated formula, it will get rid of the double negation and put the positive version of the formula in the tableau instead. This decision does not affect the soundness and completeness of this logic. It also does not make the tableau less understandable for end-users: This step is skipped very often in hand-made tableaus. On top of this, it makes the solver more time-efficient.

With the extension of extendability, there are some other things we need to consider before building the solver. The first thing is that the extendability extension can create infinite branches. This needs to be avoided if possible, as there might still be a valid way to close the branch before ending up in an infinite loop. One way of solving this is that the rule of extendability is only applied with the lowest priority, only if there are no relations available yet to further the tree, and so only when it is needed and there is no other valid option. If we do end up in an infinite branch, the program halts after a pre-determined maximum number of worlds.

When it comes to the actual coding, there are a few sub-problems to solve in order to make the eventual solver. The first part of the solver needs to parse the inference. As mentioned before, the input part of the solver needs to be user-friendly so that the user can provide the inference without the use of illogical representation symbols. This part also needs to prepare the input for the tableau-method solving. This includes separating the inferences in two categories: premises and conclusions, making the conclusions negative, and labeling them by their corresponding world. After this the solver needs to do the actual solving. The last part includes printing the resulting tableau in a user-friendly way.

Before we can solve the inference, we need to parse the user input. For this we will require the user to put in machine-readable signs, because not all logic operators exist on a standard QWERTY-keyboard. The translations of these operators are given in Table 2.2.

**Table 2.2: Translation of junctions**

| operator | input |
|---|---|
| $\neg$ | ~ |
| $\wedge$ | & |
| $\vee$ | \| |
| $\rightarrow$ | > |
| $\leftrightarrow$ | = |
| $G$, $H$, $F$ and $P$ | G, H, F and P |

The inferences are separated by commas, and the premises and conclusions are separated by a "-". The first thing the code does is making two lists, one containing the premises and the other containing the conclusion. The latter will have to be negated after parsing. The parsing itself is pretty straightforward. The function *buildParseTree* moves through the inference recursively, adding left, or right trees with every bracket. The inference uses a stack to keep track of the parent. The pseudo-code can be found in Algorithm 2.1. It can be seen in this algorithm how the double negation is handled. The variable negation can only be True or False, the function *CHANGENEGATION* changes True to False and vice versa.

**Algorithm 2.1** buildParseTree

```
tree ⇐ TREE()
stack ⇐ STACK()
stack.APPEND(tree)
while input is not empty do
  i ⇐ POPINPUT()
  if i =~ then
    tree.CHANGENEGATION()
  else if i =' (' then
    stack.APPEND(tree) tree = tree.left
  else if i = (G or F or H or P) then
    tree.SETJUNCTION(i)
    stack.APPEND(tree)
    tree = tree.left
  else if i = atom then
    tree.SETJUNCTION(i)
    parent = stack.POP()
    tree = parent
  else if i = (& or | or > or =) then
    tree.SETJUNCTION(i)
    stack.APPEND(tree)
    tree = tree.right
  else if i =')' then
    tree = stack.POP()
  end if
end while
return tree
```

Once all inferences are parsed and the conclusions are negated, the solver can come to the actual solving. The solver makes an initial branch from the list of inferences as all these inferences will be in the very top branch of the tableau. It then starts solving the branch in the function *solve*, that can be found in Algorithm 2.2. In this function, the branch is solved recursively, adding left, right and mid sub-branches when applicable. It applies the extendability rule only when it cannot find a solution for the highest priority tree (and so there is nothing else we can do). It chooses the highest priority tree by sorting the branch according to the priority scheme given in Table 2.2, and the times the tree has been chosen already. Trees that have already been chosen a few times are less likely to be chosen again.

**Algorithm 2.2** solve

```
branch.INSERTIONSORT() {sorts the branch according to priority and times used}
tree ⇐ branch.POP()
tree.used+ = 0.5
if tree.REACHEDLEAF() then
  if not branch.CLOSED() then
    SOLVE(branch)
  end if
else if branch.ISBRANCHING then
  branch1 ⇐ branch
  branch1.relations.add(arb)
  branch.left ⇐ SOLVE(branch1)
  branch2 ⇐ branch
  branch2.relations.add(a=b)
  branch.mid ⇐ SOLVE(branch2)
  branch3 ⇐ branch
  branch3.relations.add(bra)
  branch.right ⇐ SOLVE(branch3)
else
  branch1, branch2 ⇐ TABLEAUSOLVE(tree)
  {returns either 2 subbranches or 1 branch according to the tableau-rules of the tree}
  if branch1 and branch2 are empty then
    branch.relations.add(EXTENDABLE(branch)
    {add a useful relation so the branch can continue}
  else if branch1 is empty then
    branch+ = branch1
    branch ⇐ SOLVE(branch)
  else
    branch1+ = branch
    branch.left ⇐ SOLVE(branch1)
    branch2+ = branch
    branch.right ⇐ SOLVE(branch2
  end if
end if
return branch
```

The last part of the code is the so-called printing part. There are two options to choose here. For one, the tableau can be printed in the terminal. This shows the tableau from left to right instead of from bottom to top. It also shows the relations per branch and the world the trees exist in. An example of such an output is given in Listing 1.

**Listing 1: output**

```
Tableau
      relations:
      0 r 1
      branch:
      'F''p' 0
      'G'~'p' 0
      'p' 1
      ~'p' 1
      X
relations:
branch:
~('F''p'='~'G'~'p') 0
      relations:
      0 r 2
      branch:
      ~'G'~'p' 0
      ~'F''p' 0
      'p' 2
      ~'p' 2
      X
```

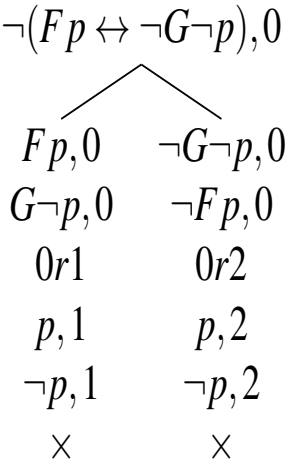
There is another option regarding the output. The code can also make a text file containing latex-style text that produces an actual tree representing the output. This tree-structure of the same output as Listing 1 is given in Figure 2.1.

This output also shows the input and whether the tableau is closed or not. When it is not closed it will show an accurate counter-model. As the output in Figure 2.1 is easier to read and more clear than the output in Listing 1, we will use this output for the rest of the paper.

## 2.2 Experimental setup

To answer the research questions, a clear setup of the experiments needs to be established. The first part of the research question requires the solver and temporal logic to be sound and complete with respect to the appropriate models. The soundness and completeness of this logic is proven in the next section. The second part

Input: $\vdash (Fp \leftrightarrow \neg G\neg p)$.
Semantic tableau:




The tableau is closed.

**Figure 2.1: Output**

of the question requires some preset temporal logic inferences that are used to compare the solver to itself including and excluding the extension of extendability. The handpicked inferences are given in Figure 2.2. The inferences are chosen in such a way that they cover all operators, and that in some inferences, namely 2, 5 and 6, extendability is required to at least get further in the tableau.

1. $a \vdash HFa$
2. $\vdash (Gp \rightarrow Fp)$
3. $\vdash (Fp \leftrightarrow \neg G\neg p)$
4. $p \vdash (Gp \vee Hp)$
5. $\vdash (FGp \rightarrow p)$
6. $\vdash PG(p \wedge \neg Hq)$

**Figure 2.2: inferences**

The inferences are all solved a 1000 times by the solver with and without the extension of extendability. The time is measured for just the solving part of the solver; the parsing and printing are only done once. The average and the standard deviation of the 1000 solves is calculated and from there the t- and p-values are calculated.

## 2.3 Soundness and Completeness

Priest (2008) has proven the soundness and completeness of $K^t$ and $K_\eta$. This proof will combine these proofs such that the soundness of $K^t_\eta$ with respect to extendable temporal models is established as well.

## 2.4 Soundness

### 2.4.1 Definition 1

*Let I = <W, R, v> be any modal interpretation (possible worlds model), and b be any branch of a tableau for* $K^t_\eta$.
*Then I is faithful to b iff there is a map, f, from the natural numbers to W such that:*

- *For every node D,i on b, D is true at world f(i) in I.*
- *If irj is on b, then f(i)Rf(j) in I*
- *If i=j is on b, then f(i) is f(j) in I*

*We say that f shows that I is faithful to b.*

### 2.4.2 Lemma 1

*Let b be any branch of a tableau, and let I = <W, R, v> be any interpretation (possible worlds model).*
*If I is faithful to branch b, and a tableau rule is applied to b, then that rule produces at least one extension b' such that I is faithful to b'.*
*If I is faithful to branch b, and i exists on branch b, then an extension b' is produced that contains irj or jri such that I is faithful to b'.*

*Proof:*
*Let f be a function which shows I to be faithful to b. Suppose that GA, i is on b, and that we apply the rule for G. Since I is faithful to b, GA is true at f(i). Moreover, for any i and j such that irj is on b, f(i)Rf(j). Hence, by truth conditions for G, A is true at f(j) for all those j, and so I is faithful to the extension of the branch. Also, suppose that FA,i is on b and we apply the rule for F to get nodes of the form irj and A,j for some j not on b. Since I is faithful to b, FA is true at f(i). Hence, for some w ∈ W, f(i)Rw and A is true at w. Let f' be the same as f except that f'(j) = w. Note that f' also shows that I is faithful to b, since f and f' differ only at j; this does not occur on b. Moreover, by definition, f'(i)Rf'(j), and A is true at f'(j). Hence, f' shows I to be faithful to the extended branch. The rule for H and P is similar.*

*For transitivity: since irj and jrk are on b, f(i)Rf(j) and f(j)Rf(k). Hence f(i)Rf(k) since R is transitive, as required.*

*For* $\eta$*: i occurs on b, and we apply the rule to get irj, where j is new. We know that for some w ∈ W, f(i)Rw. Let f' be the same as f except that f'(j) = w. Since j does not occur on b, f' shows that I is faithful to b. Moreover, f'(i)Rf'(j) by construction. Hence, f' shows that I is faithful to the extended branch. For* $\eta$*': i occurs on b, and we apply the rule to get jri, where j is new. We know that for some w ∈ W, wRf(i). Let f' be the same as f except that f'(j) = w. Since j does not occur on b, f' shows that I is faithful to b. Moreover, f'(j)Rf'(i) by construction. Hence, f' shows that I is faithful to the extended branch.*

*For* $\varphi$*: Suppose that irj and irk are on b. Then f(i)Rf(j) and f(i)Rf(k). By the forward convergence constraint, f(j)Rf(k) or f(k)Rf(j) or f(j)=f(k). So f shows at least one of the branches obtained by applying the rule to be faithful to b. For* $\beta$*: Suppose that jri and kri are on b. Then f(j)Rf(i) and f(k)Rf(i). By the forward convergence constraint, f(j)Rf(k) or f(k)Rf(j) or f(j)=f(k). So f shows at least one of the branches obtained by applying the rule to be faithful to b.*

### 2.4.3 Theorem 1

*For finite* $\Sigma$*: if* $\Sigma \vdash_{K^t_\eta} A$*, then* $\Sigma \vDash_{K^t_\eta} A$.

*Proof:*
*Suppose that* $\Sigma \nvDash_{K^t_\eta} A$*. Then there is an interpretation, I = <W, R, v>, that makes every premise from* $\Sigma$ *true, and A false, at some world, w* $\in W$*. Let f be any function such that f(0) = w. This shows I to be faithful to the initial list. The proof is now exactly the same as in the non-modal case (Priest, 2008)*

## 2.5 Completeness

### 2.5.1 Definition 2

*Let b be an open branch of a tableau for* $K^t_\eta$.
*The interpretation I = <W, R, v> that is* induced *by b is defined as follows:*

- $W = \{ w_i : i \text{ occurs on } b \}$;
- $w_i R w_j$ *iff irj occurs on b;*
- *If i=j is on b then f(i) is f(j)*
- *If* $w_i \in W$*, then for some j,* $w_i R w_j$

- *If $w_i \in W$, then for some j, $w_j R w_i$*
- *If $w_i R w_j$ and $w_j R w_k \in R$, then $w_j R w_k \in R$*
- *If p,i occurs on b, then $v_{wi}(p) = 1$;*
  *if ¬p,i occurs on b, then $v_{w_i}(p) = 0$*

#### 2.5.2 Lemma 2

*Let b be an open complete branch of a tableau.*
*Let I = <W, R, v> be the interpretation induced by b.*
*Then for all (also complex) inferences D and for all i, the following holds:*

*If D,i is on b, then $v_{w_i}(D) = 1$.*
*if ¬D,i is on b, then $v_{w_i}(D) = 0$.*
*The proof is by recursion on the complexity of A. If A is atomic, the result is true by definition. Suppose that A is of the form GB. If GB,i is on b, then for all j such that irj is on b, B,j is on b. By construction and the induction hypothesis, for all $w_j$ such that $w_i R w_j$, B is true at $w_j$. Hence, GB is true at $w_i$, as required. If ¬GA,i is on b, then F¬A,i is on b; so, for some j, irj and ¬A,j are on b. By induction hypothesis, $w_i R w_j$ and A is false at $w_j$. Hence, GA is false at $w_i$ as required. The case for F, H and P are similar.*

*For transitivity: for $w_i$, $w_j$, $w_k \in W$, suppose that $w_i R w_j$ and $w_j R w_k$. Then irj and jrk occur on b; but then irk occurs on b (by the transitivity rule). Hence, $w_i R w_k$, as required.*

*For η: if $w_i \in W$ then for some j, irj is on b. Hence, for some j, $w_i R w_j$, as required. For η': if $w_i \in W$ then for some j, jri is on b. Hence, for some j, $w_j R w_i$, as required.*

*For φ: Suppose that $w_i R w_j$ and $w_i R w_k$ (where i, j, and k are distinct). Then irj and irk are on b. Because the φ-rule has been applied, either jrk, krj, or j=k is on b; so either $w_j R w_k$ or $w_k R w_j$ or f(j) is f(k). In the last case, j=k, so $w_i = w_j$. In all three case, we therefore have what we need. For β: Suppose that $w_j R w_i$ and $w_k R w_i$ (where i, j, and k are distinct). Then jri and kri are on b. Because the φ-rule has been applied, either jrk, krj, or j=k is on b; so either $w_j R w_k$ or $w_k R w_j$ or f(j) is f(k). In the last case, j=k, so $w_i = w_j$. In all three cases, we therefore have what we need.*

#### 2.5.3 Theorem 2

*For finite Σ: if $\Sigma \models_{K^t_\eta} A$, then $\Sigma \vdash_{K^t_\eta} A$.*
*Proof:*
*Suppose that $\Sigma \not\models_{K^t_\eta} A$. Given an open branch of the tableau, the interpretation that this induces makes all the premises true at $w_0$ and A false at $w_0$ by the Completeness Lemma. Hence, $\Sigma \not\models_{K^t_\eta} A$.*

# 3 Results

The solver is able to construct a tableau for all given inferences in a reasonable time. See Table 3.1 for the average milli-seconds it took to solve the 6 inferences on average over a 1000 runs with extendability on and off. It also shows the standard deviation of the inferences, and then the t- and p-value of the two means per inference. Inferences 2, 5, and 6 require extendability in order to get further in the tableau. For inferences 1 and 3, the difference in average times is not significant according to an un-paired two-sample t-test ($t(0,780976719) = 0,214945, p > 0.5$) and ($t(0,285975574) = 0,387507, p > 0.5$). For inferences 2, 4, 5 and 6, when extendability is off, the times are significantly shorter than when extendability is on according to an un-paired two-sample t-test ($t(215,511772) > 0.00001, p < 0.05$), ($t(-5.96562) = 0,000039, p < 0.05$), ($t(404,0119734) > 0.00001, p < 0.05$) and ($t(233,9500421) > 0.00001, p < 0.05$). It can also be noted that, although the means of inference 4 are significantly different, their means are not as different as in inferences 2, 5 and 6.

**Table 3.1: Average times of the six inferences, t- and p-value**

| Formule | Extendable | Average milli-seconds | Standard Deviation | t-value | p-value |
|---|---|---|---|---|---|
| 1 | no | 0,10935987 | 0,010491327 | 0,780976719 | 0,214945 |
| | yes | 0,109703309 | 0,009127852 | | |
| 2 | no | 0,05668339 | 0,008086576 | 215,511772 | <0.00001 |
| | yes | 0,320763961 | 0,03789626 | | |
| 3 | no | 0,496430584 | 0,070329504 | 0,285975574 | 0,387507 |
| | yes | 0,497324656 | 0,069484488 | | |
| 4 | no | 2 | 0,070154793 | -5,96562 | 0,000039 |
| | yes | 2 | 0,105865634 | | |
| 5 | no | 0,196671139 | 0,032972549 | 404,0119734 | <0.00001 |
| | yes | 1 | 0,095699331 | | |
| 6 | no | 0 | 0,00160981 | 233,9500421 | <0.00001 |
| | yes | 9 | 1,161395242 | | |

In Figure 3.1 you can see how inference 6 is solved without the extension of extendability. In Figure 3.2 you can see inference 6 with the extension of extendability on, it also shows a correct counter-model it got from the left-most branch. The rest of the solved inferences can be found in Appendix A.

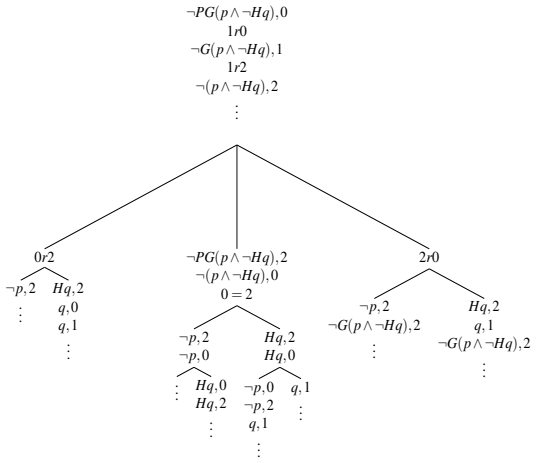
Input: $\vdash (PG(p \wedge \neg Hq))$.
Semantic tableau:

$$\neg PG(p \wedge \neg Hq), 0$$

The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_0\}$
$R = \emptyset$

**Figure 3.1: Inference 6, non-extendable**

Input: $\vdash (PG(p \wedge \neg Hq))$.
Semantic tableau:




The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_i \mid i \in Z\}$ where Z is the set of integers {..... -2, -1, 0, 1, 2, .....}
$R = \{< w_0, w_2 >, < w_1, w_2 >, < w_1, w_0 >\} \cup \{< w_i, w_1 > \mid i > 2\} \cup \{< w_2, w_j > \mid j < 0\}$
$v_{w_2}(p) = 0$

**Figure 3.2: Inference 6, extendable, countermodel read from the leftmost branch**

# 4 Discussion

## 4.1 Conclusion

Let's now look at the research questions again:

*1. Can we make a non-branching temporal linear-time logic solver that allows infinite open branches and that is sound and complete?*

*2. How fast does a non-branching temporal linear-time logic solver that allows infinite open branches solve a set of temporal linear-time logic problems compared to a normal non-branching temporal linear-time logic solver using the tableau method?*

To answer the first question: yes we can. We made a non-branching temporal linear-time logic solver that is sound and complete with respect to the relevant semantics as long as extendability is turned "on". The solver does what it is supposed to do, it solves the inferences. The soundness and completeness of this logic is proven in the Methods-section.

The second question is less straight-forward to answer. Looking at Table 3.1 in the results section, we can find the average milli-seconds it took to solve the 6 inferences on average over a 1000 runs with extendability on and off. It also shows the standard deviation of the inferences, and then the t- and p-value of the two means per inference. The inferences for which extendability was used to get further down the branch were inferences 2, 5, and 6. Looking at the p-values of these inferences, it was concluded that these inferences take significantly more time to be solved than the same inference without the extension of extendability. For the inferences that do not require extendability the situation is not as clear. Inference 4 requires significantly more time with the extension of extendability while the other two inferences (1 and 3) do not differ significantly in their time to solve the inference with and without extendability. It has to be noted, however, that although inference 4 differs significantly in its time with and without extendability, it does not differ as much as inferences 2, 5 and 6. This raises suspicion that the significance in inference 4's result might be due to something else than inferences 2, 5 and 6.

Looking at the individual times of all the different inferences, we can see that even for the same inference with either the extension off or on that the calculation times still vary very much. This might be the cause for

the significant result found for inference 4. A way to reduce the chance of this happening is creating longer inferences. With longer inferences, its takes longer to solve them so the differences in calculation time will be relatively less.

Another reason for the significant difference in 4's calculation time might lie in the inference itself. The tableau is not closed and produces a countermodel. The fact that it is not closed means the inference would try anything to get further in the tableau. The way the solver is made, it does not create a relation when it does not help to further the tableau, but that does not mean that the solver never reaches the if-statement $if extandable == True$ during the solving. So it might have been this extra if-statement that has been requested many times because the solver couldn't get any further, that created the extra time it took the solver with the extension of extendability.

In the end, the big significant results in the inferences that require extendability as opposed to the inferences that do not require extendability that either don't have a significant result or have a significant result of questionable reliability, make the conclusion quite clear. The non-branching temporal linear-time logic solver solves a set of infinite temporal linear-time logic problems that require the extension of extendability to get further in the tableau more slowly compared to these inferences without the extension of extendability.

So what does this conclusion tell us exactly? One could argue that needing more time to solve an inference is a bad thing. It also requires more computational power. In the end however, the difference in computation time between the inferences with extendability or without is minimal. The solver itself is pretty fast at solving the inferences, and even with much larger inferences the difference will hardly be noticeable. On the other hand, the extension of extendability makes sure the inference can get further down the tableau, and it might be able to solve the inference this way. The extension of extendability might make the solver an unnoticeable bit slower, it does help with a better insight in certain inferences. Additionally, without extendability, one sometimes does not obtain the correct solution.

## 4.2 Reflection

Looking at the solver's structure, there is one issue with extendability. A certain stop needed to be implemented to avoid the creation of an infinite tableau. It was already discussed in the methods section that we chose to implement a *maxWorld* variable; this would stop the solver once it wants to make the world with integer *maxWorld*, all solutions in this paper had a *maxWorld* of three. This is not a very elegant way, however. The user might not know what a good *maxWorld* variable is and using one that is too small might terminate the solver before it has found the existing counter model. If the variable is set too high, on the other hand, it might take more time and computational resources than necessary. It might also create a tableau that is not as simple as it could be and is therefore not as clear. A better solution to solving the infinity issue might be to detect reoccurring actions in the solver. Once the solver starts repeating actions, it will not find a valid countermodel anymore and the solver can be terminated. It is very hard, however, to detect these loops, as they can be of varying lenghts and also present in sub- and parent-trees. The way this solver is built, it would be hard to implement this idea.

## 4.3 Further research

For further research, there are a few things to keep in mind. As mentioned in the reflections, it might be a good idea to re-implement the handling of the infinite-worlds problem. Furthermore, looking at the conclusion, a set of longer inferences might make the results more clear and the conclusion stronger.

A way to extend the program could be to add more extensions, so that the user can see how inferences are solved with different extensions. One way to do this could be that the user can choose the extensions they want to implement. Another way, or an added feature could be that the program shows in the output for which combination of extensions the inference is solvable. This could create a more universally-applicable application.

# References


Evert Willem Beth. *Semantic Entailment and Formal Derivability*. Noord-Hollandsche Uitgeverij, 1955.

E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, apr 1986. ISSN 0164-0925. doi: 10.1145/5397.5399. URL `http://doi.acm.org/10.1145/5397.5399`.

Amy Felty and Laurent Théry. Interactive theorem proving with temporal logic. *Journal of Symbolic Computation*, 23(4):367–397, 1997. ISSN 07477171. doi: 10.1006/jsco.1996.0094.

Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. In *IEEE Transactions on Automatic Control*, volume 53, pages 287–297, 2008. ISBN 3-540-33170-0. doi: 10.1109/TAC.2007.914952.

Graham Priest. *An Introduction to Non-Classical Logic: From Ifs to Is*. Cambridge University Press, 2008.

# 5 Appendix A: Results

Input: $a \vdash (HFa)$.
Semantic tableau:

$$a, 0$$
$$\neg HFa, 0$$
$$1r0$$
$$\neg Fa, 1$$
$$\neg a, 0$$
$$\times$$

The tableau is closed.

**Figure 5.1: Inference 1, non-extendable**

Input: $a \vdash (HFa)$.
Semantic tableau:

$$a, 0$$
$$\neg HFa, 0$$
$$1r0$$
$$\neg Fa, 1$$
$$\neg a, 0$$
$$\times$$

The tableau is closed.

**Figure 5.2: Inference 1, extendable**

Input: $\vdash (Gp \rightarrow Fp)$.
Semantic tableau:

$$\neg(Gp \rightarrow Fp), 0$$
$$\neg Fp, 0$$
$$Gp, 0$$

The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_0\}$
$R = \emptyset$

**Figure 5.3: Inference 2, non-extendable**

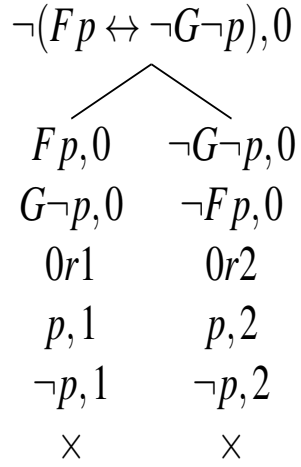
Input: $\vdash (Gp \rightarrow Fp)$.
Semantic tableau:

$$\neg(Gp \rightarrow Fp), 0$$
$$\neg Fp, 0$$
$$Gp, 0$$
$$0r1$$
$$\neg p, 1$$
$$p, 1$$
$$\times$$

The tableau is closed.

**Figure 5.4: Inference 2, extendable**

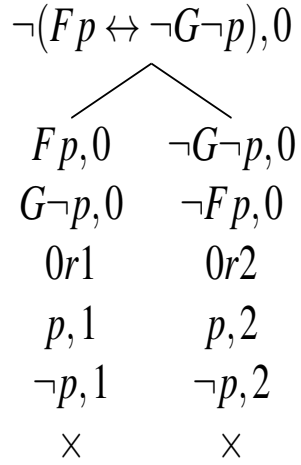
Input: $\vdash (Fp \leftrightarrow \neg G\neg p)$.
Semantic tableau:




The tableau is closed.

**Figure 5.5: Inference 3, non-extendable**

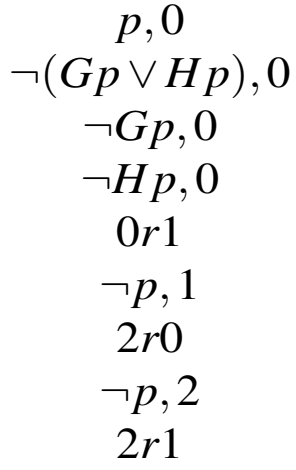
Input: $\vdash (Fp \leftrightarrow \neg G\neg p)$.
Semantic tableau:




The tableau is closed.

**Figure 5.6: Inference 3, extendable**

Input: $p \vdash (Gp \vee Hp)$.
Semantic tableau:




The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_0, w_1, w_2\}$
$R = \{< w_0, w_1 >, < w_2, w_0 >, < w_2, w_1 >\}$
$v_{w_0}(p) = 1$
$v_{w_1}(p) = 0$
$v_{w_2}(p) = 0$

**Figure 5.7: Inference 4, non-extendable**

Input: $p \vdash (Gp \vee Hp)$.
Semantic tableau:

$$\neg(Gp \vee Hp), 0$$
$$p, 0$$
$$\neg Gp, 0$$
$$\neg Hp, 0$$
$$0r1$$
$$\neg p, 1$$
$$2r0$$
$$\neg p, 2$$
$$2r1$$

The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_i \mid i \in Z\}$ where Z is the set of integers {..... -2, -1, 0, 1, 2, .....}
$R = \{< w_0, w_1 >, < w_2, w_0 >, < w_2, w_1 >\} \cup \{< w_1, w_i > \mid i > 2\} \cup \{< w_j, w_2 > \mid j < 0\}$
$v_{w_0}(p) = 1$
$v_{w_s}(p) = v_{w_1}(p) = v_{w_2}(p) = 0$, where $s$ can be any integer $s \notin \{0, 1, 2\}$

**Figure 5.8: Inference 4, extendable**

Input: $\vdash (FGp \rightarrow p)$.
Semantic tableau:

$$\neg(FGp \rightarrow p), 0$$
$$\neg p, 0$$
$$FGp, 0$$
$$0r1$$
$$Gp, 1$$

The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_0, w_1\}$
$R = \{< w_0, w_1 >\}$
$v_{w_0}(p) = 0$

**Figure 5.9: Inference 5, non-extendable**

Input: $\vdash (FGp \rightarrow p)$.
Semantic tableau:

$$\begin{array}{c} \neg(FGp \rightarrow p), 0 \\ \neg p, 0 \\ FGp, 0 \\ 0r1 \\ Gp, 1 \\ 1r2 \\ p, 2 \\ 0r2 \\ \vdots \end{array}$$

The tableau is not closed, the max number of worlds is:3
Here is the countermodel:
$W = \{w_i \mid i \in Z\}$ where Z is the set of integers {. . . . . -2, -1, 0, 1, 2, . . . . .}
$R = \{< w_i, w_j > \mid i < j\}$
$v_{w_0}(p) = 0$
$v_{w_i}(p) = 1$ for all $i \geq 1$

**Figure 5.10: Inference 5, extendable**