UNIKASSEL
VERSITÄT

Bachelor's Thesis

An Efficient Algorithm for Proof Search in the Calculus of Influence

Sören Möller

Research Group
Theoretical Computer Science / Formal Methods
Prof. Dr. Martin Lange

supervised by
Martin Lange

September 24, 2022

Statutory Declaration

I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this, nor a similar work, has been published or presented to an examination committee.
Also I assure, that the digitally submitted version of the thesis matches the printed version.

_____

Zimmersrode, September 24, 2022
Sören Möller

Contents

1. Introduction

Today, society benefits from computer systems while the industry and other areas depend on improvements made possible by these. Schools are mainly digitalized by providing electronic whiteboards, often referred to as smart boards. Additionally, tablets are used for writing notes instead of the traditional pen and paper. However besides that, no further advantages of the digitalization are used in school yet, while the usage of these digital tools could support skills and strategies that are needed in scientific contexts [4, p. 3]. Studies confirm, that methodologies using digitalized resources can improve their efficiency [9].

Since the requirement for each subject differs, there is not one tool that is sufficient for all subjects. To address this, different tools for different subject have to be created. In this case, the biology subject was picked to create a possible learning tool for it. Considering biology often concentrates on different factors influencing each other, the tool should be able to represent this behavior. Therefore, *influence experiments* (see Section 2.2) are introduced to capture these influences. The purpose of the learning tool is to read data about an experiment and to allow checking whether a hypothesis about the influence of the two underlying factors of the experiment is true. This can especially be useful when experiments cannot be done in class due to timing or safety reasons.

This thesis creates the foundation for building such a tool, by providing a calculus for checking whether a hypothesis is true (see Chapter 3) and acquiring an algorithm to check this for real world experiments. For that, in Chapter 2, the influence experiments are formally described and the internal representation of them will be introduced. After the calculus being introduced, the algorithm (see Chapter 4) for applying the proof rules will be introduced with the main goal of being efficient considering time complexity. After that, the implementation (see Chapter 5) and its design choices will be presented, while introducing useful extension modules. Finally, the implementation will be tested using benchmarks representing real world experiments, which will be scaled in granularity to see the time complexity considering more dense modeled experiments. To finalize the research done in this thesis, it will be concluded in Chapter 7.

2. Preliminaries

Before introducing the calculus which will be the foundation of the algorithm, the *influence experiments* will be introduced. Especially the *statements* will be formally defined which are used to abstract and store data from a given experiment and are used inside the calculus to check whether a hypothesis is true. While the experiments are not needed for understanding the calculus and the acquired algorithm, it is beneficial to introduce them for understanding the origin of the proof rules and for analyzing the time complexity of the algorithm for an input size sufficient for these experiments. Additionally, since the definitions rely on intervals, they are introduced separately with the used notations and properties.

2.1. Intervals

We define $\mathbb{R}_\infty$ as $\mathbb{R} \cup \{-\infty, \infty\}$ and define an order on it by giving the relation $\leq_{\mathbb{R}_\infty} = \leq_{\mathbb{R}} \cup \{(x, \infty) \mid x \in \mathbb{R}\} \cup \{(-\infty, x) \mid x \in \mathbb{R}\}$, with $\leq_{\mathbb{R}} \in \mathbb{R} \times \mathbb{R}$ being the order on $\mathbb{R}$. By that, the order uses the order of the underlying set while respecting the newly added extreme points. Analogously, this can be defined for $\mathbb{Q}_\infty$.

Definition 1 (Interval)**.** Let $a, b \in \mathbb{R}$, then the following intervals can be defined as subsets of $\mathbb{R}$:

$$
\begin{aligned}
[a, b] &= \{x \mid a \leq x \leq b\} && (\textit{closed interval}) \\
(-\infty, b] &= \{x \mid -\infty < x \leq b\} && (\textit{left-opened interval}) \\
[a, \infty) &= \{x \mid a \leq x < \infty\} && (\textit{right-opened interval}) \\
(-\infty, \infty) &= \{x \mid -\infty < x < \infty\} && (\textit{opened interval})
\end{aligned}
$$

For representability reasons, the borders a, b should be in $\mathbb{Q}_\infty$. To avoid distinction between the borders being in $\mathbb{Q}$ or in $\{-\infty, \infty\}$, always $[a, b]$ will be used, abusing the introduced notation.

Furthermore, an order on intervals is defined to be able to search through a collection of intervals more efficiently.

Definition 2 (Interval order)**.** $[a, b] \leq [x, y]$ applies for $a, b, x, y \in \mathbb{R}_\infty$, if $a < x$ or $a = b$ and $b \leq y$ holds. By that, *order* on a collection of intervals can be defined.

2.2. Influence Model

To be able to formally proof assumptions over an experiment, these experiments have to be defined and abstracted into a mathematical space. With $\mathcal{V} = \{a, b, ...\}$ being a finite set of variables, an *influence experiment* can be defined as a function $\mathcal{F} : \mathcal{V}^2 \to (\mathbb{R} \to \mathbb{R})$. On top of that, a partial order for $\mathcal{V}$ should exist that is being respected by the influences, meaning no variable should influence another variable being lower in the order than itself. Each function $\mathcal{F}(a, b)$ with $a, b \in \mathcal{V}$ describes the influence of variable a on variable b as a (partial) piece-wise function. Additionally, the influences should behave in a transitive way and variables influencing itself should behave like the identity function.

To derive provable, abstracted data from an experiment, statements are defined. These statements model a variable $a \in \mathcal{V}$ on a given interval influencing a variable $b \in \mathcal{V}$ on another interval with a quality, describing if the influence in this area follows a *monotonic* ($\nearrow$), *antitonic* ($\searrow$), *constant* ($\rightarrow$) or *arbitrary* ($\rightsquigarrow$) way.

Definition 3 ($\mathcal{V}$-Statement)**.** A $\mathcal{V}$*-statement* is a 5-tuple (a, I_1, q, I_2, b) with $a, b \in \mathcal{V}$. Furthermore, let I_1, I_2 be intervals over $\mathbb{R}_\infty$ as introduced in Section 2.1 and $q \in \{\rightsquigarrow, \searrow, \nearrow, \rightarrow\}$ be the *quality*. For readability reasons, a $\mathcal{V}$-statement is denoted as $a \xRightarrow{I_1\, q\, I_2} b$. If $\mathcal{V}$ is clear from the context, we may only speak of a statement instead of a $\mathcal{V}$-statement.

For such a statement to satisfy an influence experiment some conditions have to be met. For a given variable pair (a, b), let $\mathcal{F}(a, b)$ be an influence experiment and $a \xRightarrow{[x,y]\, q\, [x',y']} b$ with $a, b \in \mathcal{V}$, $q \in \{\searrow, \nearrow, \rightarrow\}$ be a $\mathcal{V}$-statement, then one of these cases should hold.

- $q = \nearrow$ and for all $z, z' \in [x, y]$ with $z \leq z' : \mathcal{F}(a, b)(z) \leq \mathcal{F}_{a,b}(z')$
- $q = \searrow$ and for all $z, z' \in [x, y]$ with $z \leq z' : \mathcal{F}(a, b)(z') \leq \mathcal{F}_{a,b}(z)$
- $q = \rightarrow$ and for all $z, z' \in [x, y] : \mathcal{F}_{a,b}(z') \leq \mathcal{F}(a, b)(z)$

In addition to that, for all $z \in [x, y]$, $\mathcal{F}(a, b)(z) \in [x', y']$ should be the fulfilled.

Definition 4 (Influence Model)**.** A $\mathcal{V}$*-influence model* $\mathcal{M}$ is a finite set of $\mathcal{V}$-statements. If $\mathcal{V}$ is clear from the context, we may only speak of an influence model instead of a $\mathcal{V}$-influence model.

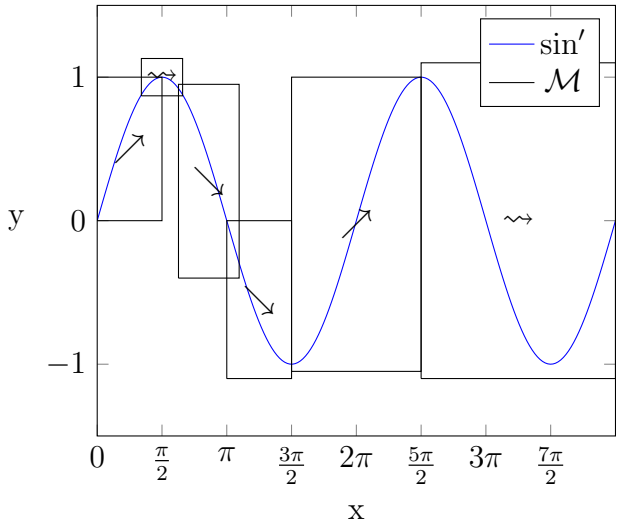


Figure 2.1.: Abstract model of $\sin'(x)$.

By that we defined a symbolic and abstract model of an influence experiment, in which proofs can be formulated. Apart from the introduced definition of an influence model, the used variables in the model can be restricted to extract a model which only included statements over a variable a influencing another variable b.

Definition 5 (Specific Model)**.** A (a, b)-model is a finite set of statements $a \xRightarrow{I\, q\, I'} b$ with I, I' being intervals and q being the quality.

Example 1. In this example the function $\sin' : [0, 4\pi] \to [-1, 1]$, $x \mapsto \sin(x)$ describes an influence experiment and is abstracted by the (x, y)-model

$$\mathcal{M} = \Big\{ x \xRightarrow{[0,\frac{\pi}{2}]\nearrow[0,1]} y,\ x \xRightarrow{[\frac{\pi-1}{2},\frac{\pi+1}{2}]\nearrow[0.87,1.13]} y,\ x \xRightarrow{[\frac{\pi}{2}-0.4,\pi+0.3]\nearrow[-0.4,0.95]} y,$$
$$x \xRightarrow{[\pi,\frac{3\pi}{2}]\nearrow[-1.1,0]} y,\ x \xRightarrow{[\frac{3\pi}{2},\frac{5\pi}{2}]\nearrow[-1.05,1]} y,\ x \xRightarrow{[\frac{5\pi}{2},4\pi]\nearrow[-1.1,1.1]} y \Big\}.$$

The function $\sin'$ and the model $\mathcal{M}$ are visualized in Fig. 2.1. The model is one out of an infinite amount of possible models that abstract the given function correctly.

As seen in Example 1 and Fig. 2.1, a (a, b)-model can be visualized in a 2-dimensional graph by plotting the statements as windows according to their intervals. To avoid distinctions these axes are called the x-axis and the y-axis, not respecting the actual variable name they represent. Consequently, the first interval of a statement is referred to as the x-interval and the second interval of the statement is referred to as the y-interval. According to this, a model can be interpreted geometrically as a set of windows with different properties.
To sort a set of statements, the order of intervals can be used, ordering the statements on their x-interval.

3. Calculus of Influence

To be able to check whether an assumption is being supported by a given model, the viable input should be defined. In the case of a $\mathcal{V}$-influence model, a $\mathcal{V}$-statement as introduced in Section 2.2 can be defined as a hypothesis. To explain the correctness of such hypothesis $\mathcal{S}$ regarding the model $\mathcal{M}$, $\mathcal{M} \models \mathcal{S}$ is defined as the behavior in $\mathcal{S}$ matching the one that is described in the model $\mathcal{M}$

To be able to automatically decide $\models$, a set of proof rules is needed, which can be used to formulate proofs. With these, the $\vdash$ relation can be decided which implies the $\models$ relation.

Definition 6. Let $\mathcal{M}$ be a $\mathcal{V}$-influence model and $\mathcal{S}$ be a $\mathcal{V}$-statement, $\mathcal{M} \vdash S$ holds if and only if $\mathcal{S}$ is derivable from $\mathcal{M}$ by using the proof rules stated in Fig. 3.1.

When referring to the process of building a proof tree to decide $\vdash$, we may refer to it as proving or solving.

To improve the intuitive understanding of the proof rules provided by Fig. 3.1 they will be discussed shortly. Furthermore, additional functions needed for the rules will be introduced.
The *Fact* rule provides a completion of a branch in the proof tree by checking whether a statement exists in the model. The *Refl* rule provides the same completion of a branch in the proof tree for a reflexive statement. For the next rules, an order on the qualities has to be defined and the term of a *stronger* statement has to be introduced.

Definition 7 (Quality order)**.** The partial order on the qualities is $\rightarrow \preceq\ q\ \preceq \rightsquigarrow$ with $q \in \{\nearrow, \searrow\}$

Definition 8 (Stronger statement)**.** A statement $a \xRightarrow{I_1\, q\, I_2} b$ is *stronger* than another statement $a \xRightarrow{I_1'\, q'\, I_2'} b$ if $I_1' \subseteq I_1, I_2 \subseteq I_2'$ and $q \preceq q'$ holds.

Moreover, the I^- rule provides the functionality to create weaker statements, which might be useful for adapting borders to be able to use the *Join*, I^+_{left} and the I^+_{right} rules. The I^+ rule on the other hand allows building the intersection window of overlapping statements. For that, the qualities of the statements will be combined,

$$\text{(Fact)}\ \frac{}{a \overset{I\,q\,I'}{\Longrightarrow} b}\ \text{if } a \overset{I\,q\,I'}{\Longrightarrow} b \in \mathcal{M} \qquad \text{(Trns)}\ \frac{a \overset{I\,q\,I_1}{\Longrightarrow} b \qquad b \overset{I_2\,q'\,I'}{\Longrightarrow} c}{a \overset{I\,q\otimes q'\,I'}{\Longrightarrow} c}\ \text{if } I_1 \subseteq I_2$$

$$\text{(I}^+\text{)}\ \frac{a \overset{I_1\,q\,I_1'}{\Longrightarrow} b \qquad a \overset{I_2\,q'\,I_2'}{\Longrightarrow} b}{a \overset{I_1\cap I_2\,\min_\preceq\{q,q'\}\,I_1'\cap I_2'}{\Longrightarrow} b} \qquad \text{(Join)}\ \frac{a \overset{[x,z]\,q\,I}{\Longrightarrow} b \qquad a \overset{[z,y]\,q'\,I'}{\Longrightarrow} b}{a \overset{[x,y]\,q\oplus q'\,I\cup I'}{\Longrightarrow} b}\ \text{if } I \cap I' \neq \emptyset$$

$$\text{(Refl)}\ \frac{}{a \overset{I\,q\,I}{\Longrightarrow} a} \qquad \text{(I}^+_{\text{right}}\text{)}\ \frac{a \overset{[x,z]\,q\,[x_0,y_0]}{\Longrightarrow} b \qquad a \overset{[z,y]\,q'\,[x_1,y_1]}{\Longrightarrow} b}{a \overset{[x,z]\,q\,[b_x(q,x_0,x_1),b_y(q,y_0,y_1)]}{\Longrightarrow} b}$$

$$\text{(I}^+_{\text{left}}\text{)}\ \frac{a \overset{[x,z]\,q\,[x_0,y_0]}{\Longrightarrow} b \qquad a \overset{[z,y]\,q'\,[x_1,y_1]}{\Longrightarrow} b}{a \overset{[z,y]\,q\,[b_x(q,x_0,x_1),b_y(q,y_0,y_1)]}{\Longrightarrow} b}$$

$$\text{(I}^-\text{)}\ \frac{a \overset{I\,q\,I'}{\Longrightarrow} b}{a \overset{I_1\,q'\,I_2}{\Longrightarrow} b}\ \text{if } I_1 \subseteq I, I' \subseteq I_2, q \preceq q'$$

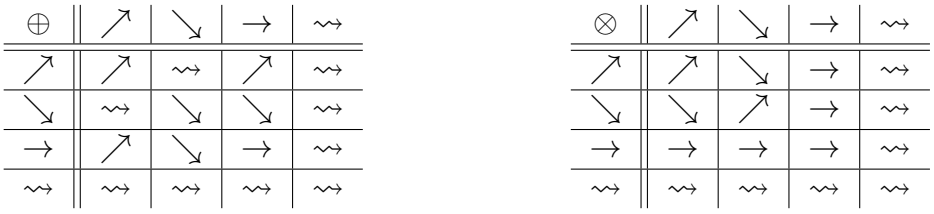
Figure 3.1.: Rules of the calculus.

taking the minimum value of the qualities of the combined statements, respecting the partial order $\preceq$.

Definition 9 (Right neighbor)**.** Statement $a \overset{[x_0,y_0]\,q\,I}{\Longrightarrow} b$ is a *right neighbor* of statement $a \overset{[x_1,y_1]\,q'\,I'}{\Longrightarrow} b$, if $x_0 = y_1$ holds.

Definition 10 (Left Neighbor)**.** Statement $a \overset{[x_0,y_0]\,q\,I}{\Longrightarrow} b$ is a *left neighbor* of statement $a \overset{[x_1,y_1]\,q'\,I'}{\Longrightarrow} b$, if $y_0 = x_1$ holds.

The I^+_{left} and I^+_{right} rules are for propagating height borders through the corresponding neighbored statements. For determining the resulting boundaries for the height, the boundary functions provided by Fig. 3.3 can be used. Next, the *Join* rule can be used on two neighbored statements, providing the functionality to create wider statements. To connect the qualities of those statements, the function $\oplus : Q \to Q$ with $Q = \{\nearrow, \searrow, \to, \rightsquigarrow\}$ can be used as stated in Fig. 3.2a. Finally, the *Trns* rule provides the functionality to propagate statements over different variables in a transitive fashion. For the combination of the qualities the function $\times : Q \to Q$ can be defined which can be extracted from Fig. 3.2b.



(a) (b)

Figure 3.2.: Operations for (a) quality add and (b) quality times.

$$b_x(q, x_0, x_1) = \begin{cases} x_0 & \text{if } q = \nearrow \\ x_1 & \text{if } q = \searrow \\ \max(x_0, x_1) & \text{if } q = \rightarrow \end{cases} \qquad b_y(q, y_0, y_1) = \begin{cases} y_1 & \text{if } q = \nearrow \\ y_0 & \text{if } q = \searrow \\ \min(y_0, y_1) & \text{if } q = \rightarrow \end{cases}$$

(a) (b)

Figure 3.3.: Bound functions for the I^+_{right} and I^+_{left} rules to determine the resulting lower (a) and upper (b) bounds.

3.1. Proofs

The proof rules of the calculus can be used to form proofs in the style of a sequent calculus [2, Sec. 1.2]. Such a proof can be represented as a tree which starts with the hypothesis as the root at the bottom, having the rules stacked on top of it to build different branches of the tree. If all the created branches end with an *axiom*, namely the *Refl* or the *Fact* rule, the proof is done and the given hypothesis $\mathcal{S}$ is derivable from the model $\mathcal{M}$. On the other hand, if such a rule cannot be applied to conclude a branch, the given proof does not imply $\mathcal{M} \vdash \mathcal{S}$. If no tree can be found fulfilling this condition, $\mathcal{M} \nvdash \mathcal{S}$ holds. Generally, a proof tree can be built in two different ways. Firstly, the hypothesis could be the starting point from which statements are build on top of it until each branch is finished with an axiom. We refer to this as the *bottom-up* approach. While this potentially creates small proof trees since no unnecessary branches are created, it is not a suitable approach in this case since the I^- rule allows the creation of an infinite amount of statements. Secondly, the initial statements of the model can be taken by using the *Fact* rule. After that, the rules of the calculus are applied to create new statements in each step with the goal of deriving the hypothesis. The approach will be referred to as the *top-down* approach.

Observation 1. For the calculus of influence, a top-down, goal driven approach is the most suitable to solve $\vdash$ using proofs.

Since the boundaries of the hypothesis are fixed, it is possible to apply a goal driven approach and avoid the abuse of the I^- rule to prevent creating an infinite amount of statements.

As stated, a given (a, b)-model can be interpreted geometrically as windows in a 2-dimensional graph. Additionally, a proof over such a model can also be intepreted geometrically, by combining the statements in an analogous manner to the proof rules. The statements of the model can be interpreted as windows and additional windows can be added to the model by using the proof rules. These equal the statements that are created by rules inside the proof tree. Those do not change the intuitive semantic interpretation of the model, since the added statements are implied by ones already in the model. With that, the proof-theoretic interpretation and geometric interpretation can be transformed into the other interpretation. The goal of the proof is to derive a stronger statement than the hypothesis. Following the geometric interpretation, the proof can be visualized by plotting the statements that are on top of the proof tree coupled to the *Fact* or *Refl* rules as windows with dashed lines. Except those, the statements that are derived by other statements using the proof rules are visualized as windows with solid lines. Lastly, the hypothesis is highlighted, having a red window. All the statements have the quality inside them, while the hypothesis is also highlighted. To improve the distinction of the bounds and qualities, the visualized model may include small offsets added to the windows to improve the explicitness of the geometric interpretation.

In the following, some examples will be discussed to clarify proofs and the rules used in it.

Example 2. Let $\mathcal{M} = \left\{ a \xRightarrow{[0.5,4.5] \searrow [0.5,3.5]} b, a \xRightarrow{[2,3] \to [2.5,3.5]} b \right\}$ be a (a, b)-model and $\mathcal{S} = a \xRightarrow{[0.5,2] \searrow [2.5,3.5]} b$ be the hypothesis. Then one possible proof tree would look like the following.

$$\dfrac{\dfrac{\dfrac{}{a \xRightarrow{[0.5,4.5] \searrow [0.5,3.5]} b}\,(\text{Fact}) \quad \dfrac{}{a \xRightarrow{[2,3] \nearrow [2.5,3.5]} b}\,(\text{Fact})}{a \xRightarrow{[2,3] \to [2.5,3.5]} b}\,(\mathrm{I}^{+}) \quad \dfrac{\dfrac{}{a \xRightarrow{[0.5,4.5] \searrow [0.5,3.5]} b}\,(\text{Fact})}{a \xRightarrow{[0.5,2] \searrow [0.5,3.5]} b}\,(\mathrm{I}^{-})}{a \xRightarrow{[0.5,2] \searrow [2.5,3.5]} b}\,(\mathrm{I}^{+}_{\text{right}})$$

Since all the paths end with the *Fact* rule, $\mathcal{M} \vdash \mathcal{S}$ is the case. In general, there can be multiple possible proof trees. In this case, we could trim the tree since the I^{+} rule is not needed for proving the hypothesis. The statements in the model are visualized by Fig. 3.4.

Example 3. Let $\mathcal{M} = \left\{ a \xRightarrow{[0.5,3] \nearrow [1.5,3]} b, a \xRightarrow{[2,4.5] \nearrow [1,2.5]} b \right\}$ be a (a, b)-model and $\mathcal{S} = a \xRightarrow{[1,4.5] \nearrow [1.25,2.75]} b$ be the hypothesis. For this, the following proof can be

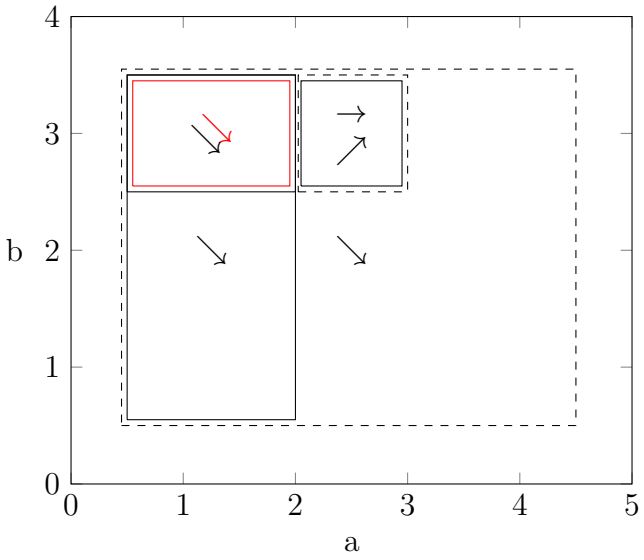


Figure 3.4.: Visualization of the proof in Example 2.

formulated. To avoid duplicates in the tree subtrees can be used on multiple occasions. For that, a label (*Label) is used whenever a subtree is build to verify that the statement on the bottom of it is actually derivable, and it is referred to with a label (Label*). When referring a subtree like that, it can be substituted it into the given position. While this notation does not work when these labels are used for different subtrees, it is sufficient for the following example.

$$\dfrac{\dfrac{\dfrac{}{a \overset{[0.5,3] \nearrow [1.5,3]}{\Longrightarrow} b}(\text{Fact}) \quad \dfrac{}{a \overset{[0.5,3] \nearrow [1.5,3]}{\Longrightarrow} b}(\text{Fact})}{a \overset{[0.5,2] \nearrow [1.5,3]}{\Longrightarrow} b}(\text{I}^-) \quad \dfrac{}{a \overset{[2,4.5] \nearrow [1,2.5]}{\Longrightarrow} b}(\text{Fact})}{a \overset{[2,3] \nearrow [1.5,2.5]}{\Longrightarrow} b}(*\text{I}^+)}{a \overset{[0.5,2] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{I}^+_{\text{right}}) \quad \dfrac{}{a \overset{[2,3] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{I}^{+*})$$

$$a \overset{[0.5,3] \nearrow [1.5,2.5]}{\Longrightarrow} b \ (*\text{Join})$$

This subtree provides a statement that we can use in the proof. Since it is needed twice, it improves the readability of the proof to extract it once and substitute it

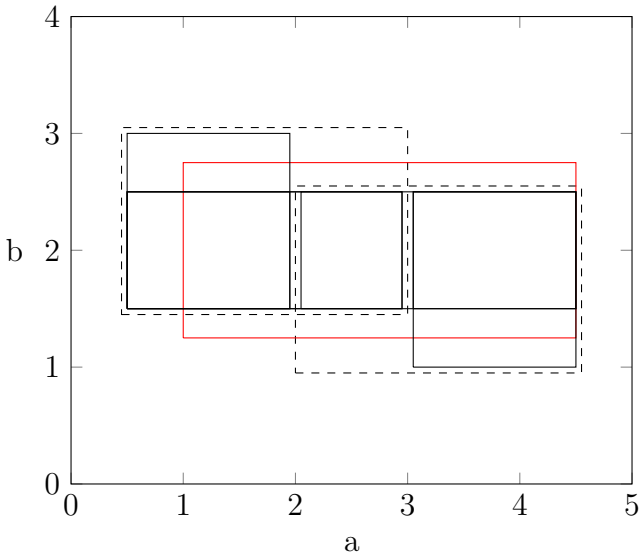


Figure 3.5.: Visualization of the proof in Example 3.

into the correct position afterwards.

$$\dfrac{\dfrac{\dfrac{}{a \overset{[0.5,3] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{Join}^*) \qquad \dfrac{\dfrac{}{a \overset{[0.5,3] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{Join}^*) \qquad \dfrac{\dfrac{}{a \overset{[2,4.5] \nearrow [1,2.5]}{\Longrightarrow} b}(\text{Fact})}{a \overset{[3,4.5] \nearrow [1,2.5]}{\Longrightarrow} b}(\text{I}^-)}{a \overset{[3,4.5] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{I}^+_{\text{left}})}{a \overset{[0.5,4.5] \nearrow [1.5,2.5]}{\Longrightarrow} b}(\text{Join})}{a \overset{[1,4.5] \nearrow [1.25,2.75]}{\Longrightarrow} b}(\text{I}^-)$$

Since the hypothesis is in the root and all paths end with the *Fact* rule, $\mathcal{M} \vdash \mathcal{S}$ holds. Again, the proof and the model is visualized in Fig. 3.5. Additionally, since all the qualities are $\nearrow$, they are not explicitly visualized.

4. Algorithm

The goal of the algorithm is to efficiently check for a hypothesis $\mathcal{S}$ and an influence model $\mathcal{M}$, whether $\mathcal{M} \vdash \mathcal{S}$ is the case. As stated, the *top-down* approach is more suitable than the *bottom-up* approach. Due to that, the algorithm has to derive new statements from $\mathcal{M}$ using the proof rules trying to derive a stronger statement than the hypothesis. Given such a statement, it can be weakened using the I^- rule to derive the hypothesis. Firstly, the problem is decidable since it is possible to build a finite cover of the model where all derivable statements are build. On that cover, it could be checked whether one of the statements is stronger than the hypothesis, which would imply $\mathcal{M} \vdash \mathcal{S}$. Although it is possible to build this cover, it has to be done with care since the I^- rule could create an infinite amount of statements. However, this approach is not sufficient since the time complexity would be too high and a goal driven algorithm is more suitable. For now, a complete and correct algorithm will be acquired, keeping the time complexity as low as possible before discussing the efficiency in Chapter 6.

4.1. Underlying Data Structure

To perform an efficient proof search, the statements should be stored in a sufficient way. For that, the access of the data has to be addressed. Since the access of statements dependents on the influence, the influence model can be split up into non-empty (a, b)-models for all influences $(a, b) \in \mathcal{V} \times \mathcal{V}$ that exist in the model. By that, the existing influences of the model can be determined and potential transitive connections in the model can be extracted more efficiently. In addition to that, statements targeting the same influence as the hypothesis $\mathcal{S}$ can be extracted. In conclusion, the model can be saved as a hash table, where $(a, b) \in \mathcal{V} \times \mathcal{V}$ are viable keys that point to a container storing the (a, b)-model. Such a container needs to support different access mechanisms to be sufficient. Namely, the random access of statements, sorting statements, querying for overlapping statements on the x-axis, checking for neighbored statements and modifying the container should be efficiently possible. Considering this, an interval tree [1, Sec. 10.1] could be used. With that,

modifying the container and randomly accessing elements is possible in logarithmic time. However, querying for overlapping intervals is not supported by the structure and intervals can only be queried for overlapping a given point. Although the query can be adjusted for querying an interval $[a, b]$ by saving all endpoints c that are included in the interval and querying for those, this process is rather inefficient since this leads to a query time of $\mathcal{O}(r \log n)$ with n being the amount of statements in the model and r being the amount of endpoints included by the query interval. This can easily extend the linear query time which can be achieved by simply iterating over all the given intervals and collecting the intervals fulfilling the overlap condition.

As a consequence of this, a list is a more sufficient data structure. The time complexity of the random access of elements is constant. While modifying the list has a linear time complexity, adding elements to the end of the list has a complexity of $\mathcal{O}(1)$. By that, the initial set of elements can be sorted in $\mathcal{O}(n \log n)$ with n being the amount of statements in the container. This is sufficient, since it can be avoided to dynamically add statements during the running time, which allows sorting the statements just once. The downside of this structure is, that querying for overlapping statements has a linear worst-case complexity. The main problem is to efficiently determine the overlapping statements of a statement that are predecessors in the sorted list. The problem will be investigated with the following example.

Example 4. Let $\mathcal{M} = \left\{ a \xrightarrow{[0,4] \to [0,1]} b, a \xrightarrow{[1,2] \to [0,1]} b, a \xrightarrow{[3,4] \to [0,1]} b \right\}$ be a sorted list of statements. For checking which statements overlap the last one in the list, this can be used as a starting point. From there, the list can be walked through to the left until a statement stops overlapping. As seen in the example, this procedure can already be stopped at the next statement. Consequently, the first statement in the list is not found while overlapping the desired area.

To address this, a *normalized* list of statements is introduced.

Definition 11 (Normalized Statements)**.** Let $L = \left\{ a \xrightarrow{[x_0,y_0]\, q_0\, I_0} b, \ldots, \xrightarrow{[x_n,y_n]\, q_n\, I_n} b \right\}$ be a list of statements. L is called *normalized*, if $\forall i \in \{0, \ldots, n-1\} : y_i = x_{i+1}$.

Consequently, a normalized list of statements does not contain statements that are enveloped by others on the x-axis except for ones where the lower bound is equal to the upper bound. Those can be removed from the list and are dealt with in Section 4.4. Consequently, problems like in Example 4 do not occur anymore. To address the initial problems the sorted list had, an algorithm for finding the overlapping statements of an area $[x, y]$ will be given. Intuitively, a sufficient entry point for the search can be determined and from there the list gets walked through to the left and right until a statement occurs that does not overlap the interval

Algorithm 1. Construct overlap map

```
1:  procedure ConstructMap(S)                    ▷ S is a set of statements
2:      points ← ∅                                              ▷ Store bounds
3:      map ← {}                          ▷ Map for intervals overlapping bounds
4:      for st in S do                                          ▷ Extract bounds
5:          points.update(st.begin, st.end)
6:          map[st.begin].add(st)
7:          map[st.end].add(st)
8:      s_points ← sorted(points)
9:      collect ← ∅
10:     for bound in s_points do                    ▷ Fix overlapping boundaries
11:         collect.update(map[s])
12:         if bound in map[s] then
13:             map[s].remove(s)
14:             collect.remove(s)
15:     return s_points, map
```

Algorithm 2. Construct normalized list

```
1:  procedure ConstructList(S)                   ▷ S is a set of statements
2:      s_points, map ← ConstructMap(S)
3:      for i in s.length - 1 do
4:          st ← intervalStr(map[s[i]])                        ▷ Use (I⁺) rule
5:          st.begin ← s[i]                                   ▷ Fix the boundaries
6:          st.end ← s[i + 1]
```

$[x, y]$. More precisely, the binary search algorithm [5, Sec. 6.2.1] can be applied to the normalized list of statements, searching for x. If the statement found by these overlaps the given area, the statements to the left and right are collected until statements occur that do not overlap the given area. Due to the sorting and the nominalization of the statements, no further statements can be collected after these occurrences and the algorithm can stop. Thereby, the running time of the algorithm is $\mathcal{O}(r + \log n)$ with n being the length of the list and r being the amount of statements overlapping $[x, y]$. Additionally, the neighbors of a statement can be determined in constant time, since they are the predecessor and successor of the statement in the list.

After extracting a sufficient data structure, an algorithm will be provided to transform a given (a, b)-model into the correct form. Firstly, as seen in Algorithm 1, the set of statements gets read and all boundaries are saved and sorted. The boundaries of the x-axis are used and can be accessed via the st.begin and st.end call on a statement object. Additionally, for each of these endpoints a set of overlapping statements is created. Initially, each statement is only associated with its bound-

aries. Since each statement does not only overlap their endpoints but rather all bounds between these endpoints, the bounds get iterated over again to fix this by collecting and inserting the statements as needed. In addition to that, the statements are removed from their endpoint since they are not needed there for the next stop of the normalization process. The ConstructMap procedure returns the sorted list of boundaries and the hash table that contains information about each boundary and the statements overlapped by it.

Observation 2. Let S be a set of statements given as an input into the ConstructMap-function. Additionally, let M be the hash table and B be the sorted list of boundaries returned by it. Now, for each boundary $b_0 \in B$ and its successor $b_1 \in B$ the set of all statement that envelop the interval $[b_0, b_1]$ is equal to $M[b_0]$.

As seen in Algorithm 2, the output of the ConstructMap is used to build the normalized list of statements. Using Observation 2, the sorted list of boundaries can be iterated over and for each bound the intersection of all the statements overlapping it can be build. For that, the bounds on the x-axis given by the current bound and the next bound in the list are taken. The min value of the qualities can be determined by reducing them pair-wise. For the bounds on the y-axis, the lowest upper bound and the highest lower bound of the statements can be taken. Thereby, the desired normalized list of statements can be extracted. Additionally, the definition requires the normalized list to have no gaps on the x-axis of the statements. If a gap in the area $[x, y]$ occurred, it can be fixed by inserting the statement $a \xrightarrow{[x,y] \rightsquigarrow [-\infty,\infty]} b$ at the correct index of the list. This fixes the problem by adding a statement that implies the behavior of the gap in the model. Since they provide no additional information, statements that share the lower and the upper bound on the x-axis can be removed from the list. This will be further investigated in Section 4.4.

Observation 3. Given a (a, b)-model and a normalized list L created by it using the ConstructList procedure, L contains all possible intersections of the statements in the (a, b)-model, leading to the qualities of each section of the x-axis being the most minimal one and not more intersection statements being buildable. The only exception for this are the statements sharing their lower and upper bound on the x-axis.

Observation 4. Given a (a, b)-model and a normalized list L created by it using the ConstructList procedure, each statement of the (a, b)-model can be reconstructed or strengthened with the statements of L and the proof rules of the calculus.

Proof. Let the (a, b)-model and L be given as stated. If a statement $s = a \xrightarrow{[x,y]\, q\, I} b$ of the (a, b)-model is not overlapped by other statements on the x-axis, it will be

in L since its bounds are only enveloped by itself. On the other hand, if s is being overlapped by other statements, the given range is generally split up depended on the bounds of the overlapping statements. Let the statements in the given range in L be $L_{[x,y]} = \left\{a \xRightarrow{[x,x_0]\, q_0\, I_0} b, a \xRightarrow{[x_0,x_1]\, q_1\, I_1} b, \ldots, a \xRightarrow{[x_n,y]\, q_n\, I_n} b\right\}$. Since the I^+ rule combines qualities by using the minimum value, $q_0 \preceq q \wedge \ldots \wedge q_n \preceq q$ holds. Additionally, for given qualities q', q'', with $q'' \preceq q'$: $\oplus(q', q'') \preceq q'$ holds. As a consequence of the I^+ rule building the intersection of the y-intervals to create the resulting y-interval, $I_0 \subseteq I \wedge \ldots \wedge I_n \subseteq I$ holds. With that, when joining these y-intervals back together, the resulting interval will remain a subset of I. Due to these facts, the statements in $L_{[x,y]}$ can be combined using the join rule, resulting in a statement $a \xRightarrow{[x,y]\, q'\, I'} b$, with $q' \preceq q$ and $I' \subseteq I$, which verifies the observation. □

4.2. Checking $\mathcal{M} \vdash \mathcal{S}$

To check $\mathcal{M} \vdash \mathcal{S}$ for a given hypothesis $\mathcal{S} = a \xRightarrow{I\, q\, I'} b$ and an influence model $\mathcal{M}$, the (a, b)-model is relevant. For now, the assumption that all information targeting this pair of variables is already in the model, meaning no more statements with these variables can be created using the transitivity rule. In Section 4.3, it is investigated how to deal with this assumption and whether it is a possible one to make.

To verify $\mathcal{M} \vdash \mathcal{S}$, a stronger statement than the hypothesis S has to be found by using the proof rules to create new statements from $\mathcal{M}$. This is the case since such a statement can be weakened using the I^- rule to match the hypothesis. That being said, the goal is to derive the most narrow statement that envelops I on the x-axis and is enveloped by I' on the y-axis. In addition to that, the statements with the strongest quality should be build. Since the quality in an area can only be strengthened by using the intersection rule on the overlapping statements and since the algorithm works on normalized lists, the model contains the strongest quality for each section of the x-interval in the model. To keep the amount of created statements as low as possible which reduces the running time of the algorithm, only the statements in the area of the hypothesis will be build. Consequently, the ConstructMap algorithm will be used to construct the hash table used to build the normalized (a, b)-model, but the statements will be created step by step rather than all at once.

Definition 12 (Overlapping)**.** A statement $a \xRightarrow{[x_0,y_0]\, q\, I} b$ *overlaps* a given area $[x_1, y_1]$ if $x_0 \leq y_1 \wedge x_1 \leq y_0$. It *conditionally overlaps* the area, when additionally $y_1 \neq x_0 \wedge y_0 \neq x_1$ holds.

Initially, the statements conditionally overlapping the interval I on the x-axis are build. By that, minimum amount of statements that combined fully envelop I on the x-axis can be extracted. If the given data does not span over I, $\mathcal{M} \not\models \mathcal{S}$ holds. Let $\mathcal{X}$ be the normalized list holding these statements. Consequently, these statements can be joined to check whether the resulting statement is stronger than the hypothesis. When joining multiple statements, for both the x and the y-axis the lowest lower bound and the greatest upper bounds of the combinded statements are used to create the intervals of the newly created statement. Let $\mathcal{T} = \left\{ a \xrightarrow{I_0\, q_0\, I_0'} b \in \mathcal{X} \mid I_0' \not\subseteq I' \right\}$ be the list of statements that conditionally overlap the hypothesis while not being enveloped by the hypothesis on the y-axis.

Generally, the height of a statement can be reduced by propagating height borders using the I^+_{right} and I^+_{left} rules. Looking into those rules, they adjust the y-interval of a statement by combining its own quality with the relative positioning of the left and right neighbor. However, whenever a statement $s \in \mathcal{T}$ with the $\rightsquigarrow$ quality occurs, the height cannot be adjusted by the rules any further, leading to the hypothesis not being derivable from the model. Whenever the y-interval of a statement in $\mathcal{T}$ is being adjusted and due to that being enveloped by the y-interval of the hypothesis, it will be removed from $\mathcal{T}$.

Observation 5. For an influence model $\mathcal{M}$ and a hypothesis $\mathcal{S}$, let $\mathcal{T}$ be created as described before. If a statement in $\mathcal{T}$ is not enveloped by the hypothesis on the y-axis while having the $\rightsquigarrow$ quality, $\mathcal{M} \not\models \mathcal{S}$ holds. With $|\mathcal{T}| = 0$, all the statements conditionally overlapped by the hypothesis are enveloped by the hypothesis on the x-axis, which enables the join of these statements to create a stronger window than the hypothesis. If $q' \preceq q$ holds for the resulting quality q' of the join, $\mathcal{M} \vdash \mathcal{S}$ holds. If not, $\mathcal{M} \not\models \mathcal{S}$ holds since as stated, the quality is already minimized due to the normalization of the statements.

This observation can be used to check whether $\mathcal{M} \vdash \mathcal{S}$ holds dependent on the statements in $\mathcal{T}$. If $|\mathcal{T}| > 0$, all statements that could potentially propagate its height borders to the statements in $\mathcal{T}$ have to be created. Generally, the statements that are not overlapping the hypothesis on the y-interval can be discarded, since the bounds of these are not in the relevant area. Since the hypothesis is known since the beginning, this can be done while reading in the statements initially. Because of this, the produced gaps in the normalized list get automatically filled in the normalization process.

To extract the minimum range of statements, the geometrically most left statement S_L and the geometrically most right statement S_R of $\mathcal{T}$ have to be extracted.

Potentially, a statements' height can be reduced using the right or the left neighbor depending on the quality and the bound of the y-interval exceeding the y-interval of the hypothesis. The direction from which a statements' height can be reduced is stated in Table 4.1. The entries of the table imply the direction from which a statement can be corrected. For that, the quality of the statement and whether the upper or the lower bound of the y-interval exceed the y-interval of the statement have to be provided. If the first statement in $\mathcal{X}$ is equal to S_L, the search direction can be checked. If the search direction does not include $\Leftarrow$, or if the first statement in $\mathcal{X}$ is not equal to S_L in the first place, $\mathcal{X}$ does not have to be extended to the left. Otherwise, the bounds of the y-interval of S_L exceeding the y-interval of S get saved. Now, the goal is to generate the minimum amount of statements that allow the correction of the bounds of S_L. To correct the upper bound, a statement is needed whose upper bound of the y-interval is included in the y-interval of the hypothesis. The lower bound can be corrected analogously. Now, $\mathcal{X}$ will be extended by repeatedly adding the left neighbor of the most left statement of $\mathcal{X}$. These can be generated by using the overlap map that was created in the normalization process. This process can be stopped in three different ways. Firstly, it can be stopped whenever the bound can potentially be corrected, since it can be propagated through the neighbors to correct the bound of S_L. Analogously, this can be done when both bounds have to be corrected, while not one statement has to potentially correcting both bounds at once. Using this, the search has to be continued until both bounds can be potentially corrected. Secondly, whenever a statement having the $\rightsquigarrow$ quality is generated which cannot correct the bounds as described, the process can be stopped since no information can be propagated through this statement. Lastly, if no more statements can be generated since the end of the data is reached, this process terminates as well.

Analogously, this process can be adapted for searching statements for correcting the bounds of the y-interval of S_R. Before that, it will be checked if $\mathcal{X}$ has to be extended by adding right neighbors, which can be determined using S_R and checking whether it matches the last statement of $\mathcal{X}$ and the checking search direction table, similarly to the left direction.

Observation 6. By extending $\mathcal{X}$ stepwise as described, the minimal amount of statements is added. The correctness of this process remains, since while more statement would lead to a potentially lower y-interval of the bounds to correct, more precision is not needed as long the bounds lie in the y-interval of the hypothesis. By that, no more statements have to be generated.

For the extracted list of statements $\mathcal{X}$, the right and left strengthening rules can

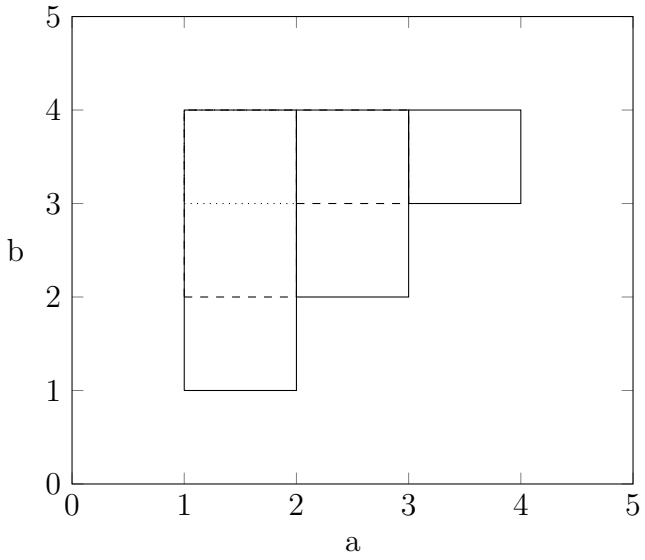


Figure 4.1.: Example of multiple usages of the I^+_{left} and I^+_{right} rules.

be applied. The following example shows, why it is not enough to strengthen each statement with its left and right neighbors once.

Example 5. Let $\mathcal{M} = \left\{ a \overset{[1,2]\searrow[1,4]}{\Longrightarrow} b, a \overset{[2,3]\searrow[2,4]}{\Longrightarrow} b, a \overset{[3,4]\searrow[3,4]}{\Longrightarrow} b \right\}$ be a set of statements, which are visualized in Fig. 4.1. Since the quality is $\searrow$ for each statement, it is omitted in the graph. By trying to produce stronger statements using the I^+_{left} and I^+_{right} rules, we may start at the most left statement in the order. On the current statement, the rules are applied respecting the left and right neighbors. Continuing this until the last statement in the list is reached, the height of the statements can be reduced as shown by the dashed lines. By reapplying this algorithm to the newly created statements, the height can be reduced even further, as shown by the dotted line. This example shows that the list of normalized statements cannot be iterated over once to minimize the height of each statement. Obviously, the algorithm cannot go through the list in reversed order either, since a counterexample for this case can be produced in an analogous manner.

Since each usage of the I^+_{left} and I^+_{right} leads to the potential improvement of its neighbors as well, statements have to be processed multiple times. An algorithm implementing this idea can be seen in Algorithm 3. To minimize the height of each statement, the list of normalized statements gets walked over from left to right and each statement is strengthened by using the I^+_{left} and I^+_{right} rules if possible. Whenever one of these rules produces a new statement, it is an existing one with its height being reduced. Due to that, the old statement can be removed, and the new one can be inserted at the given index. Whenever a statement is successfully

Algorithm 3. Strengthen interval height

```
1: procedure StrengthenIntervals(L)
2:     i ← 0
3:     while i < L.length do
4:         changed ← ⊥
5:         if i > 0 then
6:             L[i] ← I⁺_left(L[i − 1], L[i])
7:             changed ← ⊤
8:         if i < L.length - 1 then
9:             L[i] ← I⁺_right(L[i], L[i + 1])
10:            changed ← ⊤
11:        if changed then
12:            i ← i − 1
13:            continue
14:        i ← i + 1
```

Table 4.1.: Search direction for using the I^+_{left} and I^+_{right} rules.

	upper bound	lower bound
↗	$\Leftarrow$	$\Rightarrow$
↘	$\Rightarrow$	$\Leftarrow$
→	$\Rightarrow, \Leftarrow$	$\Rightarrow, \Leftarrow$

strengthened, the left neighbor has to be checked again. For this reason, bounds can be propagated back to the left end of the list. When the statement could not be strengthened, the algorithm continues with the next statement in the list. This is repeatedly done until the end of the list is reached.

Observation 7. For a normalized list of statements L, the procedure StrengthenIntervals modifies the list in such way, that each statement has its minimum height, meaning that no rule can reduce its height even further.

This algorithm can be applied to the extracted list of statements $\mathcal{X}$. Additionally, the modified statements in $\mathcal{T}$ are removed if the given invariant for a statement in the list does not hold anymore. Afterwards, $|\mathcal{T}|$ can be used to decide $\mathcal{M} \vdash \mathcal{S}$ using Observation 5.

4.3. Building the Transitive Cover

For a given influence model $\mathcal{M}$ and a given hypothesis $\mathcal{S} = a \xrightarrow{I\,q\,I'} b$, the relation $\mathcal{M} \vdash \mathcal{S}$ can be decided in its corresponding (a, b)-model. This is done under the

assumption of all the statements creatable by the use of the transitivity rule already being in the model. Since generally, this is not always the case, a way of creating the *transitive cover* over all the used variables will be presented, including all statements needed to potentially decide $\mathcal{M} \vdash \mathcal{S}$.

Definition 13 (Dependency Graph)**.** Let $G = (V, E)$ be a directed, acyclic graph. For a given $\mathcal{V}$-influence model $\mathcal{M}$, let $V = \mathcal{V}$. Since each node represents a variable, the terms node and variable are used as synonyms regarding the graph. Additionally, let E contain all the variable pairs $(a, b) \in \mathcal{V} \times \mathcal{V}$ with a non-empty (a, b)-model extractable from $\mathcal{M}$.

By that, a graph is defined, where each edge (c, d) represents a (c, d)-model. This graph has two main purposes. Firstly, while adding statements to the model and building the graph, it can be checked whether a partial order on the variables exist. For this, the acyclic behavior of the graph has to be verified. In case of the graph not being acyclic, no partial order on the set of variables exists. To check this, the depth first search algorithm [3, Sec. 22.3] can be used while memorizing the visited variables to report whenever a variable is visited multiple times, which indicates a circle in the graph. Additionally, the graph can be used to check which models are relevant for building statements using the transitivity rule to propagate all the statements into the (a, b)-model. Considering this, the depth first search algorithm can be used as well by starting at a and memorizing all variables on the current path, collecting them whenever b is reached from that path. After going through all possible paths, the variables are extracted that lie between a and b. Now, all the other variables and the associated edges can be removed from the graph.

Observation 8. For checking $\mathcal{M} \vdash \mathcal{S}$ for a given influence model $\mathcal{M}$ and a given hypothesis $\mathcal{S} = a \overset{I\,q\,I'}{\Longrightarrow} b$, only the models represented by edges that lie between a and b in the generated dependency graph are relevant for solving.

Consequently, a graph $G = (V, E)$ is created which only contains the variables that are relevant for propagating statements into the (a, b)-model in a transitive manner. The goal is to reduce this graph, until only the variables a and b are left over. To do this, a variable $v \in V \setminus \{a, b\}$ is picked and for each predecessor $pre \in V$ and each successor $succ \in V$ of v, the transitivity rule is applied to combine the statements of the (pre, v)-model with the statements of the $(v, succ)$-model, to create statements for the $(pre, succ)$-model. If this model is not existent yet, the edge will be added to the graph and the corresponding model will be created to put the created statements in its container. After all combinations of predecessors and successors are processed, v and its associated edges can be removed from the graph.

Generally, the order of the variables being reduced does not influence the efficiency of the approach. But since the goal is to create a statement stronger than the hypothesis $a \xrightarrow{I\,q\,I'} b$, its intervals can be used to determine which statements have to be build. Looking at the transitivity rule, when successfully applying the rule to a statement S' and a statement S'', the x-interval of the resulting statement equals the one of S' and the y-interval of the resulting statement equals the one of S''. Considering this, the model can be split up into different categories. Let $M_0 = \{(a, c) \in E \setminus \{(a, b)\} \mid c \in V\}$ be the set of models that involve a, additionally let $M_1 = \{(c, a) \in E \setminus \{(a, b)\} \mid c \in V\}$ be the set of models that involve b and let $M_2 = E \setminus (M_0 \cup M_1 \cup \{(a, b)\})$ be the remaining models. Lastly, let m be the (a, b)-model. Since for deciding $\mathcal{M} \vdash \mathcal{S}$, only statements which overlap the y-interval of $\mathcal{S}$ are used for solving as seen in Section 4.2, all statements not overlapping the y-interval of each model in M_1 can be removed. By lowering the amount of statements in these models, the search space for building transitive statements involving these models is reduced. To use this, an order of reducing the variables of the graph is extracted, which ensures that the models in M_1 are used in every step.

Definition 14 (Reducing Order)**.** For a dependency graph $G = (V, E)$ and a hypothesis $a \xrightarrow{I\,q\,I'} b$, let the reducing order on the variables $V \setminus \{a, b\}$ be a partial order given by the distance of a variable $v \in V \setminus \{a, b\}$ to the variable b, with the distance being the amount of edges between these variables.

To actually extract this order out of a graph, the breadth first search algorithm [3, Sec. 22.2] can be used, starting from b and collecting all variables encountered, except a and b. An example can be seen in Fig. 4.2, where a dependency graph is visualized with the order noted as labels on the nodes. The variables of the hypothesis are highlighting as a starting and an ending node to emphasizes their importance. In addition to that, the graph is visualized after the first variable in the order is reduced. Until now it is declared in which order the transitive connections are build. In the following, the most efficient way of building the transitive statements will be discussed.

In general, the use of the transitivity rule should be reduced to the minimum to improve the running time of the algorithm. Furthermore, all the important statements have to be built for the correctness of the algorithm. Before trying to minimize the use of the transitivity rule while extracting which statements are important for solving, it is discussed why previous work has to be done to ensure the completeness of the transitive cover. Unfortunately, it is not possible to build the transitive cover first before using other rules to combine and strengthen statements. The reasons for that are discussed in the following.

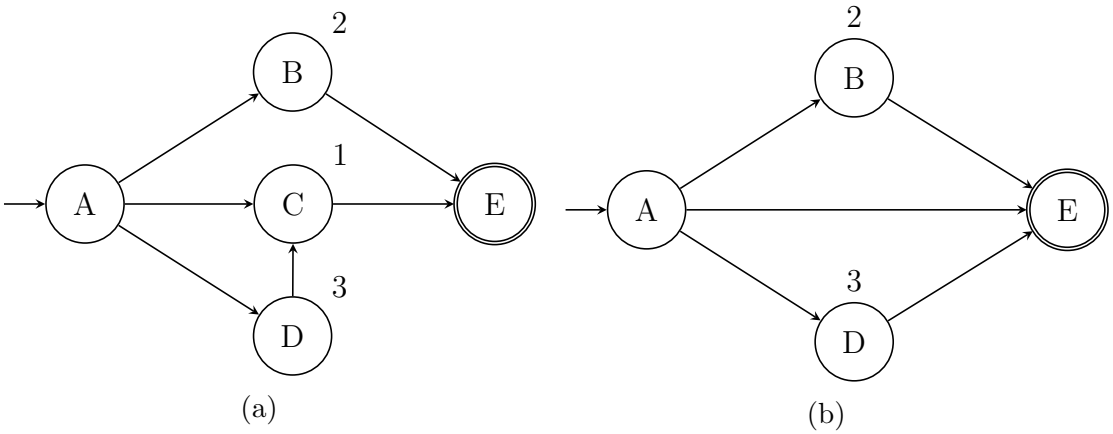


Figure 4.2.: Dependency Graph annotated with the reducing order (a) and the graph after the first variable is reduced (b).

Observation 9. Given a proof for a model $\mathcal{M}$ and a hypothesis $\mathcal{S}$, the proof tree is not modifiable in such way, that the transitivity rule is used directly after the *Fact* rule.

Proof. Let $\mathcal{M} = \left\{ b \xRightarrow{I_1\, q'\, I_1'} c, b \xRightarrow{I_2\, q'\, I_2'} c, a \xRightarrow{I_3\, q\, I_3'} b \right\}$ be an influence model, then a subtree could be the following.

$$\dfrac{\dfrac{\dfrac{}{a \xRightarrow{I_3\, q\, I_3'} b}\,(\text{Fact}) \qquad \dfrac{\dfrac{}{b \xRightarrow{I_1\, q'\, I_1'} c}\,(\text{Fact}) \qquad \dfrac{}{b \xRightarrow{I_2\, q'\, I_2'} c}\,(\text{Fact})}{b \xRightarrow{I_1 \cap I_2\, q'\, I_1' \cap I_2'} c}\,(I^+)}{a \xRightarrow{I_3\, q \otimes q'\, I_1' \cap I_2'} c}\,(\text{Trns})$$

If this tree is being reconstructed into using the *Trns* rule first, the following tree can be extracted.

$$\dfrac{\dfrac{\dfrac{}{a \xRightarrow{I_3\, q\, I_3'} b}\,(\text{Fact}) \qquad \dfrac{}{b \xRightarrow{I_1\, q'\, I_1'} c}\,(\text{Fact})}{a \xRightarrow{I_3\, q \otimes q'\, I_1'} c}\,(\text{Trns}) \qquad \dfrac{\dfrac{}{a \xRightarrow{I_3\, q\, I_3'} b}\,(\text{Fact}) \qquad \dfrac{}{b \xRightarrow{I_2\, q'\, I_2'} c}\,(\text{Fact})}{a \xRightarrow{I_3\, q \otimes q'\, I_2'} c}\,(\text{Trns})}{a \xRightarrow{I_3\, q \otimes q'\, I_1' \cap I_2'} c}\,(I^+)$$

While the result initially looks the same, this tree may be incorrect since the rules are not applied correctly. Looking at the transitivity rule, a statement can be connected with another one in a transitive way, if the variables match and if the y-interval of the first statement is a subset of the x-interval of the second statement.

Due to this, we can extract $I_3' \subseteq I_1 \cap I_2$, since the transitivity rule applied. In the reconstructed tree, the relations $I_3' \subseteq I_1$ and $I_3' \subseteq I_2$ are needed. Since a counterexample is constructable where $I_3' \subseteq I_1 \cap I_2$ holds but $I_3' \subseteq I_1$ or $I_3' \subseteq I_2$ does not, the transitivity rule cannot be pushed to the top of the proof tree. □

To ensure not missing important statements in the process of building the transitive cover, the models have to be prepared to allow all possible transitive statements to be build. Considering that, the *Trns* rule will be investigated. No statements have to be joined before building the transitive cover, since this behavior can always be reconstructed in the final (a, b)-model. As mentioned, when applying the transitivity rule, the y-interval of the first statement has to be a subset of the x-interval of the second statement. Since the statements of the models in $M_0 \cup M_2$ are the ones used as the first component in the transitivity rule, the height of these statements has to be minimized. For that, the models are normalized and Algorithm 3 is used to minimize the height of their statements.

Given a statement $a \xRightarrow{I_0\, q\, I_0'} c$, the transitivity rule can be applied for each statement $c \xRightarrow{I_1\, q\, I_1'} b$ of the (c, b)-model, with $I_0' \subseteq I_1$. Since the width of statements can be enlarged by joining multiple statements together, all statements conditionally overlapping I_0' of the (c, b)-model are extracted and joined. If the join was successful, the resulting statement can be used to apply the transitivity rule.

To solve a hypothesis $a \xRightarrow{[x,y]\, q\, I'} b$ the relevant statements can be generated step by step to lower the amount of total statements as seen in Section 4.2. This idea can also be implemented when applying the transitivity rule. This only works for the models in M_0, since here the information about the x-interval of the hypothesis can be used. Given a model $(a, c) \in M_0$, all statements conditionally overlapping $[x, y]$ are generated, before creating transitive statements using them. If a statement is generated that overlaps x, it is checked whether it is enveloped by I' on the y-axis. If this is not the case, the left neighbors of those conditionally overlapping $[x, y]$ are generated repeatedly while checking if a statement can be built using the transitivity rule that fixes this bound as described in Section 4.2. When such a statement is found, the process terminates remembering the lower bound of the last created statement on the x-axis as x_{low}. If the statement overlapping x is enveloped by I' on the y-axis anyway, set $x_{\text{low}} = x$. Analogously, this is done for the statement overlapping y where the bound is saved as x_{high}. When others models of M_0 are processed, statements conditionally overlapping $[x, y]$ are generated analogously, but the search that just has been described can be canceled when exceeding the interval $[x_{\text{low}}, x_{\text{high}}]$.

To conclude the method of building the transitives, it is given by Algorithm 4.

Algorithm 4. Build transitive cover

procedure BUILDTRANSITIVECOVER($\mathcal{M}$, $a \xRightarrow{[x,y]\,q\,I} b$)
  $G = (V, E) \leftarrow$ buildGraph($\mathcal{M}$)
  Extract the reducing order $[v_0, \ldots, v_n]$
  Split $\mathcal{M}$ into M_0, M_1, M_2 and m
  Normalize and minimize height of models in M_0 and M_2
  Remove statements of models in M_1 not overlapping I on the y-axis
  $x_{\text{low}} \leftarrow -\infty$, $x_{\text{high}} \leftarrow \infty$
  for v **in** $[v_0, \ldots, v_n]$ **do**
    for pre **in** getPredeccesors(G, v) **do**
      for suc **in** getSuccesors(G, v) **do**
        Build transitive statements as described, distinguishing whether
        $(pre, v) \in M_0$ or $(pre, v) \in M_2$ holds.
        In case of $(pre, v) \in M_0$, update the x_{low} and x_{high}
        values to avoid creating statements exceeding them.

4.4. Special Cases

In case of the hypothesis being a reflexive statement, the introduced algorithm does not have to be applied, rather a simpler algorithm can be used. This is the case, since the influence of a variable on itself is the identity function $I : \mathbb{R} \to \mathbb{R}, x \mapsto x$, which can always be derived and does not have to be added to the model. By that, the hypothesis is false, if the quality is neither $\nearrow$ nor $\rightsquigarrow$. A given reflexive hypothesis $a \xRightarrow{[z,z']\,q\,I'} a$ is true, whenever the identity function intersects the left and right border of its window. More precisely, the hypothesis is true whenever $I(z) \in I'$ and $I(z') \in I'$ holds. When receiving a reflexive hypothesis, the algorithm can do this check since the previously presented algorithm only works on a non-reflexive hypothesis. It is easily verifiable that this geometric interpretation of checking a reflexive hypothesis is recreatable using the proof rules, by combining the *Refl* rule and the I^- rule.

When creating a normalized list of statements, the intersection of neighbored statements is not done which would lead to statements where the lower and upper bound of the x-axis are equal. Normally, this is not a problem since can be applied creating statements which can be weakened using the I^- rule to match the width. But due to the intersection rule, the quality might be strengthened to $\rightarrow$. By that, information could be lost by not building the intersection like this. This special case can be evaded, by building the intersection if the lower and upper bound of the x-axis of the hypothesis are equal. Additionally, a rule for changing the quality

of such a statement to the quality $\rightarrow$ could be added to the proof rules, since these always fulfill the definition of a statement having the $\rightarrow$ quality.

Lastly, the model that is being received by the algorithm should be consistent.

Definition 15. A model $\mathcal{M}$ is *consistent*, when for each overlapping statement, the y-intervals of the statements also overlap. Additionally, no inconsistencies should be creatable using the proof rules.

Since to be able to check this, all possible statements derivable using the transitivity rule have to be created, the algorithm expects the input to already be consistent. If not, many optimizations targeting the minimization of the amount of built statements using the transitivity rule could not be used.

4.5. Completed Algorithm

Since all parts for checking whether a hypothesis $\mathcal{S} = a \overset{I\,q\,I'}{\Longrightarrow} b$ is derivable by a model $\mathcal{M}$ are acquired, they are collected and put into order to complete the algorithm.

1. Split up the model $\mathcal{M}$ into all possible non-empty (c, d)-models.
2. Check whether the hypothesis is true in the (a, b)-model. Thereby, build the statements conditionally overlapping I and extend the resulting list if needed to propagate height borders as described in Section 4.2. Additionally, initialize the x_{low} and x_{high} values and set them if possible.
3. If $\mathcal{M} \vdash \mathcal{S}$ could not be positively decided yet, build the transitives using the BuildTransitiveCover procedure, using the already initialized values of x_{low} and x_{high}. Else, return $\mathcal{M} \vdash \mathcal{S}$.
4. Finally, try solving in the (a, b)-model again reporting whether $\mathcal{M} \vdash \mathcal{S}$ or $\mathcal{M} \nvdash \mathcal{S}$ is the case.

4.6. Complexity and Correctness

For the complexity of the algorithm, the main question is whether the algorithm runs in a polynomial time complexity. Intuitively, the running time of the algorithm should not exceed this, since no new bounds are created during the process, only statements are created with bounds already existent in the model. The only rule

potentially threatening this, is the I^- rule. However, since it is only used in the normalization process, only bounds already existent in the model will be used to trim the statements. While this implies a polynomial time complexity, the complexity will be analyzed further by going through the important steps of the algorithm, estimating the complexity regarding the input size.

Firstly, let n be the average amount of statements in each (a, b)-model. Additionally, let k be the amount of (a, b)-models extractable from the given model $\mathcal{M}$. To normalize each (a, b)-model, all bounds have to be extracted and sorted, before the statements between these bounds are build and sorted. The maximum amount of bounds extracted will be $2n = m$. Let l be the average amount of statements overlapped by each statement, which is relevant when building the intersection statement between the bounds. In total, m statements are created. This leads to the normalization algorithm running in $\mathcal{O}(k * (n \log n + m * l))$

For a hypothesis $a \xRightarrow{I\,q\,I'} b$ spanning the whole (a, b)-model in the width, the whole model has to be generated. The time complexity of this process was already described in the normalization algorithm. To minimize the statements of the model in height, Algorithm 3 is used. In the worst case of the correction of each statement enabling the correction of all other statements, the algorithm runs in a quadratic time. Afterwards, the join of the statement can be created in linear time, leading to a combined time complexity of $\mathcal{O}((m)^2 + m)$.

Lastly, Algorithm 4 is analyzed to conclude the time complexity of the creation off the transitive cover. The time complexity of operations on the dependency graph $G = (V, E)$ is $\mathcal{O}(|V| + |E|)$ for the depth first search and breadth first search algorithms. Since the amount of variables is low compared to the amount of statements in the model, the impact of the graph on the running time is low anyway. When building the transitive cover, a maximum amount of $k - 1$ pairs of (a, b)-models can be used to create new statements using the transitivity rule. To create new statements for the (a, c)-model using the (a, b)-model and the (b, c)-model, For each statement of the (a, b)-model the overlapping statements in the (b, c)-model have to be searched. While the query time is logarithmic in average cases, in the worst case of all statements in the (b, c)-model overlapping the query range, it is linear. Since the statements returned by the query have to be joined together, they have to be traversed a second time. This leads to the construction of the transitive cover having a complexity of $\mathcal{O}((k - 1) * m * 2m)$.

Observation 10. Since all the acquired time complexities are polynomial and the different procedures are executed sequentially, the time complexity of the algorithm is polynomial.

For the algorithm to be correct, it has to be sound and complete. In order to be sound, whenever the algorithm finds a statement S_0 implying $\mathcal{M} \vdash \mathcal{S}$, S_0 has to derivable from model $\mathcal{M}$. The soundness of the algorithm is given, since only the rules of the calculus are used to create new statements and the rules are sound. For the algorithm to be complete, it should never return $\mathcal{M} \nvdash \mathcal{S}$ when a proof can be given which would imply the opposite. For that it has to be shown that all possible proof trees can be modified to match ones that are implied by the algorithm. Since, for example, the algorithm never uses the I^- rule without using the intersection first, some work has to be done to show each proof tree can be modified to use the I^- rule like that. Since this exceeds the purpose of this thesis, it will be left for future work.

5. Implementation

In general, the implementation strongly follows the algorithm described in Chapter 4. It was implemented in Python for reasons described in Section 5.1. Since the algorithm is already described, the language specific implementation will not be explained, rather the usage of the solver and the folder structure of the project will be described to facilitate possible modifications. In addition to that, some more specific implementation details will be discussed. Finally, a tool for plotting the statements in the model and a tool for building influence models from experiments will be described.

5.1. Choice of Programming Language

The goal of this thesis is to develop an algorithm that is capable of efficiently checking if a given statement is derivable from a given model $\mathcal{M}$. To be able to implement the algorithm and check its performance on real world test cases and benchmarks, the used programming language has to be chosen first. In this case, Python was chosen.

Python [6] is a general-purpose programming language which is dynamically-typed and emphasizes a simple syntax. By that, the language provides implementations that have a good readability and a low development time. Since it is an interpreted language, it does not get compiled and safety checks are done while executing the program. This leads to the language being rather slow compared to compiled languages like C.

Python was chosen, since its advantages concerning the development time and the readability of the algorithm fit the purpose of the implementation. To improve the understandability of the implementation even further, type-hinting was used to visualize the types of the variables to the reader. It should be noted that since Python is rather slow, there would be faster implementations possible in other languages. But since in this thesis the traceability is important, Python is the best option. Because of the usage of new features, python version 3.10 has to be used to run the algorithm.

5.2. Project Structure

To improve the accessibility of the around 1750 lines of code, the folder structure is presented and the purpose of each folder will be explained. The implementation can be found on GitHub[1].

```
influence_solver
├── benchmark
│   ├── benchmark.py
│   ├── csv_to_model.py
│   └── transitive.py
├── data
│   ├── altitude_pressure.csv
│   └── angle_intensity.csv
├── examples
│   ├── current.py
│   ├── intersect.py
│   └── other.py
├── statementstruct
│   ├── statement.py
│   ├── statement_list_dynamic.py
│   ├── statement_list_static.py
│   ├── overlap_map.py
│   └── util.py
├── plotter
│   ├── plotter.py
│   └── images.png
├── solver
│   ├── constants.py
│   ├── dependecy_graph.py
│   ├── rules.py
│   ├── solver.py
│   └── util.py
└── main.py
```

Firstly, the *benchmark* folder contains code used to perform a benchmark and measure the timing of different experiments each performed with an increasing amount of statements. This will be explained further in Chapter 6. Namely, the *benchmark.py*

[1] `https://github.com/SoerenMoeller/influence_solver.git`

file can be executed to perform the benchmark. Through adapting the parameters in the file the increase of the statements per step and the amount of steps can be adjusted. The file *csv_to_model.py* provides the functionality to extract a parameterized model out of an experiment and will be further explained in Section 5.5. Lastly, the *transitive.py* file contains a function to create a parameterized model to provide a benchmark that can be scaled over the size of the transitivity graph.

Secondly, the *data* folder contains experiments that are saved as csv-files. The format of these files will be explained in Section 5.5.

Thirdly, the *examples* folder contains different models that can be used for debugging or testing different parts of the algorithm.

Fourthly, the *intervalstruct* folder contains the data structures used in the algorithm, where the different sets of models M_0, M_1 and M_2 and the model m as introduced in Section 4.3 are split up into different data structures according to the procedures applied to them. Generally, all of them manage normalized lists of statements.

The *plotter* folder contains code providing the functionality of plotting the model which may be used for debugging. For plotting the qualities, the symbols for those are included as images. For each image a red counterpart exists to enable highlighting a statement. This will be further explained in Section 5.4.

Finally, the *solver* folder contains the actual solver, which will be explained in Section 5.3. The *solver.py* file contains the solver class which acts as the interface with the user and contains the main solving algorithm. The *constants.py* file defines some constants and the *rules.py* file contains functions which implement the rules of the calculus. While the *util.py* file provides useful functionality that may be used in other files, the *dependency_graph.py* file contains the dependency graph and the algorithms applied to it.

5.3. Usage of the Solver

The solver can be accessed by initiating an object of the solver class. To add the statements of an influence model $\mathcal{M}$, different methods are provided. Initially, a container or a single statement can be given as an argument when initiating the solver object to add these statements. Additionally, the *add* method can be used to add statements in the same way. By that, a model can be dynamically build and added to the solver object instead of collecting them all once and then adding them. To match a set-like interface, the *discard* and *remove* methods provide the option to remove statements from the solver object. Finally, the *solve* method can be used on

the solver object. It expects the hypothesis $\mathcal{S}$ as a statement and returns a boolean value indicating if $\mathcal{M} \vdash \mathcal{S}$ holds.

The statements in the model and the hypothesis are expected as 5-tuples matching the ones in Definition 3. An example of the usage can be seen in *main.py*.

5.4. Debug Tools

When using the solver, some extra information may be expected by the user instead of just getting a boolean returned indicating the result. This information will be printed to the console. For that, the *verbose* flag was introduced, having 3 different levels to choose from.

Case verbose= 0: No additional information will be printed (standard)

Case verbose= 1: Timing information will be printed. More specific, the time of adding the relevant statements to the solver, the initial solving time in the final model, the time of building the transitive cover and the final solving time in the final model will be printed. Additionally, the total solving time will be presented. In addition to that, the amount of initially added statements to the solver and the final amount of statements will be compared. Finally, the result will be printed.

Case verbose= 2: Additional windows occur, showing the plot of the initial model and the model after the proof.

To activate the flag, the *_verbose* flag of the solver object can be changed, or it can be changed using the *v* parameter of the solve function. For plotting the statements currently in the model, a procedure was written to take these models and plot them using *matplotlib* according to the geometric interpretation. The window of each statement in the model is visualized having black solid borders, while the hypothesis is highlighted in red.

To be able to use this tool, *matplotlib* has to be installed which can be done using the following command in the console.

```
$ pip install matplotlib
```

5.5. CSV Reader

Typically, influence models are abstractions of influence experiments. Such experiments are most likely given as measurement data. By that, experiments can be

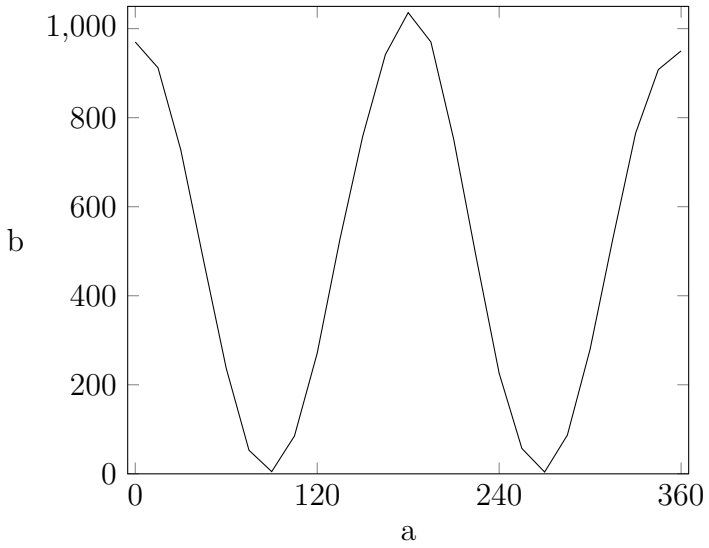


Figure 5.1.: Influence experiment of Table A.1.

seen as tables with two columns. The entries of the first row determine the variable names. Consequently, a table with a, b as the entries in the first row, can be used to create a (a, b)-model. For that, the other entries of the lower rows are points in the 2-dimensional vector space. These tables can be stored in csv files to store experiment data. These can be read and formed into a model with statements. One file can hold multiple influences, by putting the tables side by side. In the following, the data will be transformed into the correct form.

For a variable pair (a, b), the data of the corresponding csv file can be read and the n data points can be collected in a list $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ which is sorted by the x_i value. Before building the statements, the gaps between the given points have to be filled by connecting neighbored points. For that, the given data points can be transformed into the following piece-wise function $\mathcal{E} : [x_1, x_n] \to [min(y_1, \ldots, y_n), max(y_1, \ldots, y_n)]$ which is an influence experiment.

$$\mathcal{E}(x) = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} * x + y_2 - x_2 * \frac{y_2 - y_1}{x_2 - x_1} & \text{if } x_1 \leq x < x_2 \\ \vdots & \vdots \\ \frac{y_n - y_{n-1}}{x_n - x_{n-1}} * x + y_n - x_n * \frac{y_n - y_{n-1}}{x_n - x_{n-1}} & \text{if } x_{n-1} \leq x \leq x_n \end{cases}$$

As shown in Fig. 5.1, the experiment captured in Table A.1 can be turned into an influence experiment as described. Now, a parameterized function is provided to manipulate the granularity of the influence model created by the experiment. There are three parameters provided, w determining the width of each statement, h determining the height of each statement and $o \in (0, 1)$ determining the relative

overlap of a statement to its neighbors.

To extract a (a, b)-model, the lowest lower bound $x_{\min} = x_1$ will be taken as a starting point. From there, a statement of the width w and height h will be created. The center of the statement will be $(x_{\min} + \frac{w}{2}, \mathcal{E}(x_{\min} + \frac{w}{2}))$. The quality of the statement can be determined by checking the pitch of the linear function of $\mathcal{E}$ for the given range $[x_0, x_0 + w]$. If this area spans over multiple cases of $\mathcal{E}$, all the pitches of the sufficient linear functions can be considered and transformed into qualities which can be added according to $\bigoplus$. After that, the next statement can be created similarly, with the x-axis bounds being shifted by $o * w$. This process will be continued until the highest upper bound x_n is exceeded.

Depending on the height and width of the statements and the pitch of the underlying experiment, neighbored statements might not overlap on the y-axis. To prevent this, it can be checked by looping over the list again and adjusting the height to the required extend if necessary.

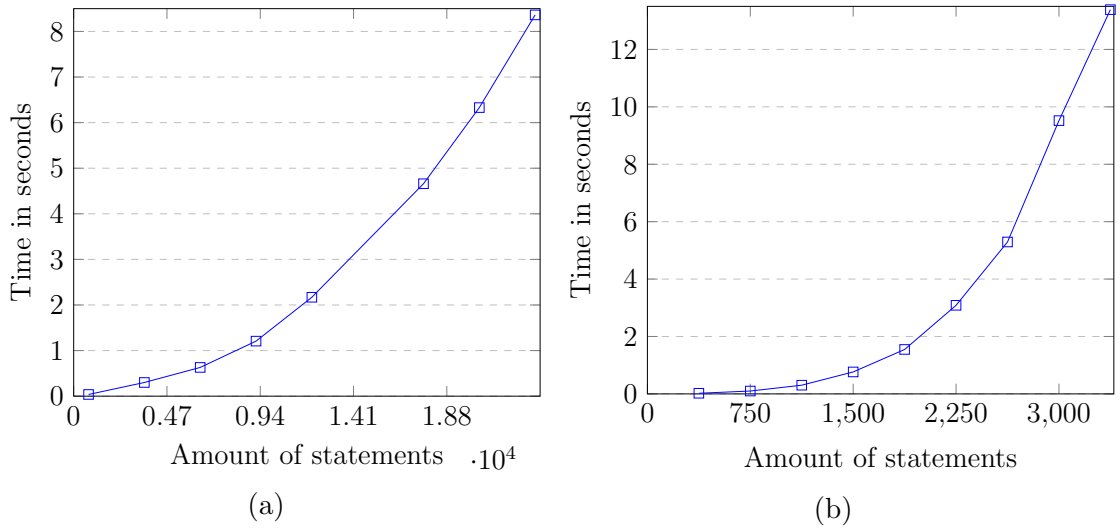
To create a model from a csv file, the *build_model_from_csv* method can be used, which expects the file name of the csv as a parameter. Additionally, the width is not given as a parameter, rather the amount of statements that should be in the model is given which can be transformed into the w variable as needed. This is more sufficient for scaling the models to benchmark the time complexity, since the amount of statements can be calculated by the amount of models that fit into the space and the overlap that each statement has.

6. Benchmark

To test the running time behavior of the implemented algorithm, some benchmarks are provided. To discuss the efficiency of the algorithm, two real world experiments will be modeled and scaled in granularity. Furthermore, the running time behavior of a benchmark scaling the amount of transitive steps taken for solving will be analyzed, since arguably the creation of the transitive cover is one of the most time-consuming steps in the algorithm. Each of the following benchmarks was run three times and the average timing for each of the configurations was used.

6.1. (Angle, Light intensity) Benchmark

Firstly, an experiment about light rays hitting a surface will be discussed. The experiment shows the relation between the angle of the light source relative to the surface area and the light intensity that can be measured on the surface. The data about this experiment can be found in Table A.1. The CSV-Reader tool was used to build a model out of the given data, which is stored in the *angle_intensity.csv* file. The first measurement was done with 750 statements being in the model. In total, 10 measurements are done while the amount of statements being increased by 2812 statements in each step. The hypothesis is being ″Angle $\xrightarrow{[0,360] \rightsquigarrow [-50,1100]}$ Light intensity″. While this ensures that the complete model has to be built, since it spans over the whole collected data, some optimizations of the algorithm cannot be used. Namely, the building of the area overlapping the x-interval of the hypothesis to minimize the amount of statements builds and the initial removal of the statements that do not overlap the y-interval of the hypothesis can not obtain any improvements, since the whole model has to be built anyway. Although, this prevents distorting the relation between the amount of statements in the model and the running time, since all the statements are effectively used. The result of the benchmark can be seen in Fig. 6.1a. There, the relation between the amount of statements and the running time is visualized. Optically, the resulting curve resembles a function in either $\mathcal{O}(n \log n)$ or $\mathcal{O}(n^2)$ with n being the amount of statements in the model. In a real world scenario, the model would most likely have a lower solving time, since



(a) (b)

Figure 6.1.: Result of the benchmarks (a) (angle, light intensity) and (b) (altitude, barometer, atmospheric pressure).

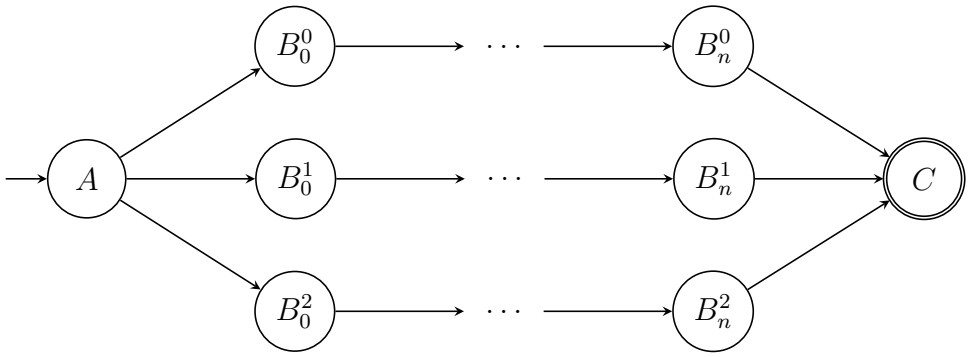
the hypothesis generally does not span the whole model.

6.2. (Altitude, Barometer, Atmospheric Pressure) Benchmark

The next influence experiment describes the influence of the altitude to barometric pressure, which itself influences the atmospheric pressure. By that, this model contains data about an experiment including a transitive step, which is given by Table A.2. Analogously to the previous benchmark, the CSV-Reader is used to build the model out of the given data. The first measurement is done with 375 statements, increasing by the same amount in each of the 10 steps. The hypothesis ″Altitude $\xRightarrow{[-100,15100] \searrow [-10,122]}$ Atmospheric pressure″ is being used, which forces the complete model to be built. Analogously to the previous benchmark, some optimizations like the lowering of the amount of transitive statements being build by using information about the x-axis and y-axis of the hypothesis cannot be used, since the hypothesis spans all the collected data. The results of this benchmark can be seen in Fig. 6.1b. Optically, the pitch of the resulting graph is higher than the previous one. That being said, the assumption that building transitives is the most heavy operations is supported. The increase of running time is due to the fact, that for building statements using the transitivity rule, all statements of the model used in the front side of the rule have to be iterated over while scanning for overlapping statements in the other model, which is a lot of effort especially in Python.

6.3. Transitivity Benchmark

To check the running time behavior for an increasing amount of variables in the model, which increases the amount of transitive steps that have to be performed for solving, no real world experiment is suitable. Consequently, an abstract benchmark was designed to check this behavior nonetheless. For that, the following dependency graph can be generated, with (A, C) being the influence which is targeted by the hypothesis and B_i^j being the variables between those.



Given such a graph, it can be scaled according to the i and j parameters to adjust the size of the graph. In the benchmark, j is set to three analogously to the graph shown above. To fill the models represented by the edges, we have to keep in mind that the transitivity rule follows a pattern that could either lead to inconsistent model or to a model that gets simpler after reducing the variables during the solving process by restricting the search space. To prevent this, all statements are enveloped by a predefined area and for all models, each point of this area is overlapped by at least one statement. The statements quality can either be $\nearrow$ or $\searrow$. To clarify this, both possibilities for the given area $[0, 5]$ are given by Fig. 6.2. Due to the $\otimes$ operation, the qualities may change during the process when using the version with the $\searrow$ quality. Coupled to the intervals, an inconsistent model could be created. Due to that, each model is build equally to Fig. 6.2b. To intensify the benchmark, the statements can be adjusted to overlap each other, to force the use of the I^+ rule. This behavior can be controlled using parameters.

In Fig. 6.3 the result of the benchmark is visualized. Here, the benchmark is scaled through the amount of transitive steps in each path. Again, 10 iterations are done starting with 10 transitive steps and increasing by 4 transitive steps each iteration. By that, the amount of statements increases linearly to the amount of transitive steps performed, starting with 33000 statements and increasing by 12000 each step. The resulting graph is more linear than the previous one, meaning the

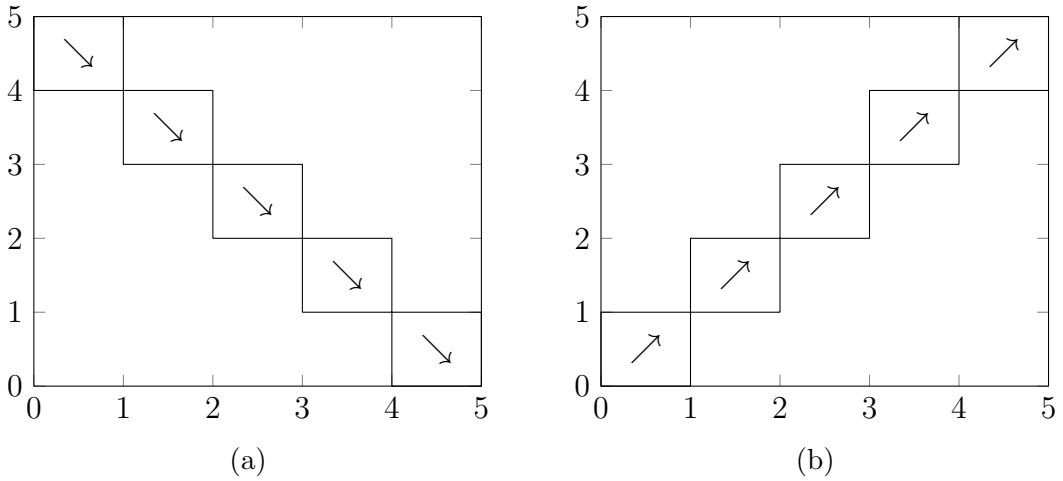


Figure 6.2.: Possible models in transitivity benchmark (a) using the ↘ quality (b) using the ↗ quality.

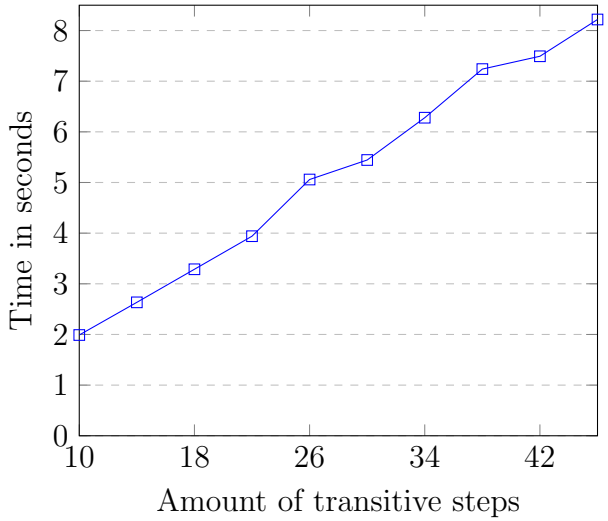


Figure 6.3.: Result of the transitivity benchmark.

benchmark does not scale negatively with the amount of transitive steps when the models in those steps are modeled with the same granularity.

6.4. Evaluation

Generally, the benchmarks support the time complexity analyzed in Section 4.6. Whether this is efficient or not is dependent on the use case. Since generally, this solver is intended to be used as a part of a learning tool, the results should be calculated in a short period of time. To set an arbitrary border, we say a solving time of up to one second is acceptable, since it could be hidden behind a transition effect in the tool.

Looking at the benchmark, especially the ones derived from real world experiments are interesting for evaluating the efficiency of the solver. Since the (angle, light intensity) benchmark firstly exceeds the 1-second mark at around 6375 statements, the algorithm can be declared as efficient in this case. The reasons for that are, that the area of the model on the x-axis $[0, 360]$ is being split up into 6375 statements. Consequently, each statement is unnecessarily slim. Additionally, each model is being generated by a fixed amount of data points. In this case, the model is extracted using 25 data points. Arguably, this number can be multiplied by a low amount to create a model which is sufficient. Besides that, the granularity of the statements should not be too high, since this could lead to deviations of the experiment not being modeled correctly or statements not being provable due to slight offsets. The same argumentation applies to the (altitude, barometer, atmospheric pressure) benchmark, even though the running time is slightly higher as already stated. This applies to all models where transitive statements have to be build. As seen in the transitivity benchmark, the amount of transitive steps does not have a heavy impact on the running time, rather the granularity and size of the underlying models. As described, the running time generally is lower than the ones measured in the benchmarks, since a hypothesis covering only parts of the collected data greatly reduces the amount of statements in the model. Concluding this, the algorithm performs an efficient proof search regarding the solving time needed for real-world experiments.

7. Conclusion

In this thesis, an algorithm for proving whether $\mathcal{M} \vdash \mathcal{S}$ holds for a given influence model $\mathcal{M}$ and a hypothesis $\mathcal{S}$ is acquired. The theoretical foundation for the proof is the calculus, which provides the rules used to extract a proof. As elaborated in Chapter 6, the algorithm performs appropriate running times for the given use case, which can be verified by the given benchmarks.

By that, the goal of the thesis to write an algorithm that expects an unmodified model and a hypothesis and solves it by using all the proof rules of the calculus using an efficient, goal driven approach is successfully reached. In practice, generally more assumptions about the model can be done to decrease the solving time of the algorithm even further. For that, the solving can be split up in two parts. Firstly, the model gets prepared by normalizing each (a, b)-model and by minimizing each statements' height as low as possible. Additionally, all possible statements are build using the transitivity rule. Afterwards, the modified model can be saved in a dedicated format, for example *json*. The second part of the algorithm would perform the proof search without having to normalize the models or having to build transitive connections. By that, only the (a, b)-model relevant for solving the hypothesis has to be read from the save-files and the statements conditionally overlapping the hypothesis can be joined together, trying to proof the hypothesis. While this would greatly reduce the running time of the solving algorithm, the first step would have an increased running time due to the whole model being prepared. But since in practice, the model will be build once and hypothesis are checked separately, this approach is sufficient. To optimize this process, the work of this thesis can be used. Additionally, the consistency of the model could be verified using this procedure. Finally, parallelization could be used to decrease the running time of this procedure even further, since multiple transitive connections could be build in parallel without interfering each other. To conclude the work, the correctness of the algorithm and the proof rules has to be proven to allow correctly checking whether a hypothesis is derivable by an influence experiment.

Bibliography

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry - Algorithms and Applications*. Springer Science & Business Media, 3rd edition, 2008.

[2] Samuel R. Buss. *Handbook of Proof Theory*. Elsevier, Amsterdam, 1998.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, 3rd edition, 2009.

[4] Delia Hillmayr, Lisa Ziernwald, Frank Reinhold, Sarah Hofer, and Kristina Reiss. The potential of digital tools to enhance mathematics and science learning in secondary schools: A context-specific meta-analysis. *Computers & Education*, 153:103897, 04 2020.

[5] Donald Ervin Knuth. *The Art of Computer Programming*. Addison-Wesley, Amsterdam, 3rd edition, 1997.

[6] Python's documentation. `https://www.python.org/doc/`. Accessed: 2022-08-01.

[7] S. Rasulzade and M. Lange. The calculus of influence for biological experiments (draft). *In preparation*, 2022.

[8] Walter Strampp. *Höhere Mathematik 2 - Analysis*. Springer-Verlag, 5th edition, 2021.

[9] Sumatokhin, Sergey, Petrova, Oksana, Serovayskaya, Delina, and Chistiakov, Fedor. Digitalization of school biological education: Problems and solutions. *SHS Web Conf.*, 79:01016, 2020.

A. Experiment Data

A.1. (Angle, Light intensity) Experiment Data

Table A.1.: Data about the (angle, light intensity) experiment.

Angle, ϕ (degrees)	Light intensity (arb. units)
0	970
15	912
30	728
45	480
60	237
75	53
90	5
105	85
120	271
135	527
150	758
165	942
180	1036
195	970
210	752
225	484
240	226
255	57
270	4
285	87
300	280
315	528
330	765
345	908
360	950

A.2. (Altitude, Barometer, Atmospheric Pressure) Experiment Data

Table A.2.: Data about the (altitude, barometer, atmospheric pressure) experiment.

Altitude (meter)	Barometer (mm Hg)	Atmospheric pressure (kPa)
-1524	903.7	120.5
-1372	889	118.5
-1219	874.3	116.5
-1067	859.5	114.6
-914	845.1	112.7
-762	830.6	110.7
-610	816.4	108.8
-457	802.1	106.9
-305	787.9	105
-152	773.9	103.1
0	760	101.3
152	746.3	99.49
305	733	97.63
457	719.6	95.91
610	706.6	94.19
762	693.9	92.46
914	681.2	90.81
1067	668.8	89.15
1219	656.3	87.49
1372	644.4	85.91
1524	632.5	84.33
1829	609.3	81.22
2134	586.7	78.19
2438	564.6	75.22
2743	543.3	72.4
3048	522.7	69.64
4572	429	57.16
6096	349.5	46.61
7620	282.4	37.65
9144	226.1	30.13
10668	179.3	23.93
12192	141.2	18.82
13716	111.1	14.82
15240	87.5	11.65
16764	68.9	9.17
18288	54.2	7.024
21336	33.7	4.49
24384	21	2.8
27432	13.2	1.76
30480	8.36	1.12