Incompleteness of the Inference System BNeg

Kjetil Midtgarden Golid

Master Thesis

Department of Informatics
University of Bergen

June 2016

Acknowledgements

I would like to thank my supervisor, Michał Walicki, for outstanding support and guidance. This thesis would not have been possible without your expertise.

Abstract

Any propositional discourse can be represented as a propositional theory in a specific form in such a way that the theory is inconsistent if and only if the discourse is paradoxical. Propositional theories in this form can be represented as directed graphs such that the models of the theory correspond to the kernels of the digraph. This thesis looks at Neg; a sound, refutationally complete, non-explosive resolution system over such propositional theories. We investigate the relation between various graph structures and clauses provable by the resolution system from the corresponding theory. The main results is a counter-example to a conjecture that a restricted version BNeg of the system is refutationally complete.

Contents

Chapter 1

Introduction and preliminaries

1.1 Paradoxes

A formula in propositional logic is semantically *inconsistent* if it has no model, i.e. there exists no variable assignment making the formula true. Likewise, a collection of formulae, called a theory, is inconsistent if there exists no variable assignment making all the formulae in the theory true. Consider the following formula:

$$x \land \neg x \tag{1.1}$$

While a sentence like $\neg a \to b$ can be satisfied by, for instance, letting both a and b be true, no such assignment can be made for the formula above. The formula is therefore inconsistent.

A paradox can be informally defined as *"a statement that can be neither true nor false"*. In this case, a statement can be both a formula or a theory. We can immediately note that since no paradoxes can be true, all paradoxes are, by definition, inconsistent. It is however not the case that all inconsistencies are paradoxes; just consider $x \land \neg x$ again: this formula simply seems false, and not paradoxical.

A different view is that paradoxes are *dialetheia* – statements that are *both* true and false[1]. We will however not spend much time exploring these philosophical differences, as this is not a philosophical paper and it will not change much for our definitions.

The liar sentence is probably the most famous example of a paradox:

> "This sentence is false."

If the sentence is true, then the sentence is false, but if the sentence is false, then the sentence is true. It can thus neither be true nor false, since both lead to a contradiction.

Notice how the liar sentence is a statement about other statements (in this case itself). A collection of formulae where some of them may refer to themselves or other formulae is called a *discourse / discourse theory* in [2], which we will follow. In order to represent such discourses, we need a formal way of referring to other statements within a statement. In propositional logic, this can be done by giving statements names. We name a statement by introducing a bi-implication between it and a fresh variable.

Consider the examples below. On the left side are normal propositional statements; on the right side are their corresponding *named* statements, the fresh variables being their names.

$$a \qquad\qquad x_1 \leftrightarrow a \tag{1.2}$$

$$a \wedge \neg a \qquad\qquad x_2 \leftrightarrow a \wedge \neg a \tag{1.3}$$

$$a \vee \neg a \qquad\qquad x_3 \leftrightarrow a \vee \neg a \tag{1.4}$$

Labelling statements in this way obviously changes their truth value. Even though there is one consistent, one inconsistent and one tautological statement on the left, all the statements on the right are consistent. This is because we can find truth values for both x_1, x_2 and x_3 that match the truth value of their corresponding statements, making each equivalence true. In other words, the truth value of a labelled statement does not refer to whether the unnamed statement is consistent, but whether or not a truth value can be found for it at all. Since we have defined a paradox to be a statement that is neither true nor false, we get that a statement is paradoxical if and only if labelling it makes it inconsistent.

Consider the liar sentence again. Labelling it and then using its label in order to make it reference itself gives us the following statement:

$$x \leftrightarrow \neg x \tag{1.5}$$

This labelled statement is obviously inconsistent, making it a paradox by our definition.

We continue to look at labelled formulae and ways of determining whether or not they are consistent.

1.2 Graph Normal Form

A propositional theory over a set of variables Σ is in *graph normal form (GNF)*[3] if all its formulae have the following form:

$$x \leftrightarrow \bigwedge_{y \in I_x} \neg y \tag{1.6}$$

where $I_x \subseteq \Sigma$ and such that every variable occurs exactly once on the left of $\leftrightarrow$ across all the formulae in the theory.

There is a simple translation from a theory in conjunctive normal form to an equisatisfiable theory in graph normal form (shown in Appendix A.1). Since conjunctive normal form is expressively complete, it follows that any propositional theory, including our labelled statements, has an equisatisfiable GNF theory.

This means that any discourse can be represented with a GNF theory such that the discourse is paradoxical if and only if the GNF theory is inconsistent. This GNF representation of discourses is interesting to us because GNF theories have a tight correspondence to graphs. This correspondence lets us not only decide the satisfiability of a discourse theory by looking at certain features in the corresponding graph, but the graph also provides us with the actual models of the discourse, if they exist.

In order to express this logic/graph correspondence, we first need to establish some graph terminology.

1.3 Kernels and Solutions

Definition 1. A *directed graph (digraph)* is a pair $\mathbf{G} = \langle G, N \rangle$ where G is a set of vertices while $N \subseteq G \times G$ is a binary relation representing the edges in $\mathbf{G}$.

We say that the graph is *finite* if its set of vertices is finite; *infinite* otherwise.

We use the notation $N(x)$ to denote the set of all vertices that are targeted by edges originating in x (*successors* of x). Similarly, $N^{-}(x)$ denotes the set of all vertices with edges targeting x (*predecessors* of x). We define the two functions formally as follows:

$$N(x) := \{y \mid (x, y) \in N\} \tag{1.7}$$
$$N^{-}(x) := \{y \mid (y, x) \in N\} \tag{1.8}$$

Definition 2. The number of successors a vertex has is often called the *out-degree* of that vertex. The number of predecessors is called the *in-degree.* If a vertex has out-degree 0, we call it a *sink*; if it has in-degree 0, we call it a *source.* If the out-degree of each vertex in a graph is finite, we say that the graph is *finitary.* A graph that is not finitary is called *infinitary.*

Note the difference between the concept of a finite graph and the one of a finitary graph; an infinitary graph is infinite, but a finitary graph is not necessarily finite.

Definition 3. A *simple path* is a sequence of distinct vertices $x_1, x_2, \ldots, x_n$ such that for each consecutive pair x_i, x_{i+1} from the sequence, we have $(x_i, x_{i+1}) \in N$. We say that two paths are *disjoint* if they do not share any vertices (with the possible exception of their initial vertices).

The number of edges in the path is called the *length* of the path.

The functions N and N^- can be extended pointwise to sets in the following way:

$$N(X) = \bigcup_{x \in X} N(x) \tag{1.9}$$

$$N^-(X) = \bigcup_{x \in X} N^-(x) \tag{1.10}$$

Definition 4. A *kernel* is a set of vertices $K \subseteq G$ such that:

$$G \setminus K = N^-(K) \tag{1.11}$$

The equivalence in the above definition can be split into two inclusions to be more easily understood:

$G \setminus K \subseteq N^-(K)$, saying that each vertex outside the kernel has an edge into the kernel (K is *absorbing*). A consequence of this is that a kernel has to be non-empty, unless the graph is empty.

$N^-(K) \subseteq G \setminus K$, saying that each edge targeting a vertex within the kernel has to come from outside, thus no two vertices in the kernel are connected by an edge (K is *independent*).

Kernels have been studied over several decades, not only in graph theory, but also within the fields of game theory and economics. The concept was first defined and used by von Neumann and Morgenstern in [4].

In a graph representing some sort of a turn-based game, where vertices are states and edges are transitions between states, one can often work out winning strategies whenever one finds a kernel in the graph. Whenever one is outside of the kernel, one always has the possibility of moving inside the kernel (since the kernel is absorbing), while inside the kernel one *has* to move out of it (since the kernel is independent). If you are the player with the choice outside the kernel, you can control the game and choose to stabilize it by always moving into the kernel, forcing the opponent to move out again on the next turn.

Deciding whether kernels exist in finite graphs has been shown to be an NP-complete problem[5]. This should not be surprising, since we are in the middle of showing the equivalence between this problem and the problem of finding satisfying models of PL-theories (SAT), which we know is NP-complete. It should be noted however, that in this thesis, we will mainly concern ourselves with the existence of kernels in *infinitary* graphs and thus also SAT over infinitary formulae.

We will get the correspondence between models of a discourse theory and kernels in a graph through an alternative, equivalent kernel definition called a *solution*.

Definition 5. Given a directed graph $\mathbf{G} = \langle G, N \rangle$, an *assignment* $\alpha \in 2^G$ is a function mapping every vertex in the graph to either 0 or 1. A *solution* is an assignment α such that for all $x \in G$:

$$\alpha(x) = 1 \iff \alpha(N(x)) = \{0\} \tag{1.12}$$

This means that for any vertex x, if x is assigned 1, then all its successors have to be assigned 0, and if x is assigned 0, then there has to exist a vertex assigned 1 among its successors. A consequence of this definition is that all sink vertices (vertices with no outgoing edges) in the graph have to be assigned 1, since it vacuously does not point to any node assigned 1. We use the notation $sol(\mathbf{G})$ to denote the set of all solutions of the graph $\mathbf{G}$.

We get the equivalence between kernels and solutions from the following two facts: Given a solution, the set of all vertices assigned 1 is a kernel. Given a kernel, the function assigning 1 to all vertices in the kernel and 0 to the rest, is a solution.

1.4 Discourse Theories and Digraphs

As mentioned earlier, there is a close connection between the following three concepts:

1. Models of a discourse (theory)
2. Kernels in a graph
3. Solutions of a graph

While we have the equivalence between (2) and (3), we will now look at two functions connecting (1) and (2). This correspondence was shown by Roy T. Cook in [6]. Let $mod(T)$ denote the set of models of the theory T; we get the following definitions from [3].

$\mathcal{T}$: translating a digraph $\mathbf{G}$ into a corresponding theory $\mathcal{T}(\mathbf{G})$ such that $sol(\mathbf{G}) = mod(\mathcal{T}(\mathbf{G}))$.

$\mathcal{G}$: translating a theory T into a corresponding digraph $\mathcal{G}(T)$ such that $mod(T) = sol(\mathcal{G}(T))$.

Given any digraph $\mathbf{G}$ we get the discourse theory $\mathcal{T}(\mathbf{G})$ by, for each vertex $x \in G$, forming the equivalence $x \leftrightarrow \bigwedge_{y \in N(x)} \neg y$. For the cases where $N(x) = \varnothing$, instead of adding an equivalence, we simply add x.

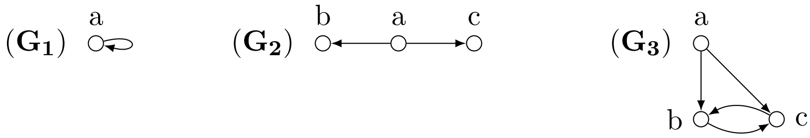

Figure 1.1

The graphs in Figure 1.1 have the following theories:

$$\mathcal{T}(\mathbf{G_1}) = \{a \leftrightarrow \neg a\} \tag{1.13}$$
$$\mathcal{T}(\mathbf{G_2}) = \{a \leftrightarrow (\neg b \wedge \neg c), b, c\} \tag{1.14}$$
$$\mathcal{T}(\mathbf{G_3}) = \{a \leftrightarrow (\neg b \wedge \neg c), b \leftrightarrow \neg c, c \leftrightarrow \neg b\} \tag{1.15}$$

The fact that $sol(\mathbf{G}) = mod(\mathcal{T}(G))$ is shown in [3]. Although not proving it, we observe that $\mathbf{G_1}$ has no solution, just like its corresponding theory $\mathcal{T}(\mathbf{G_1})$ has no models. $\mathbf{G_2}$ has one solution, where one assigns $a = 0, b = 1, c = 1$. This assignment also works as the only model for $\mathcal{T}(\mathbf{G_2})$. In $\mathbf{G_3}$, we get two solutions, both with a assigned 0, but with 0 and 1 distributed between b and c. These are also the only two models of $\mathcal{T}(\mathbf{G_3})$.

Conversely, given any discourse theory T (in fact, this will work given any PL theory, since we can translate CNF to GNF), we can derive the corresponding graph $\mathcal{G}(T)$ in the following way: All variables in the theory are vertices, and for each formula $x \leftrightarrow \bigwedge_{y \in I_x} \neg y$ make a directed edge $\langle x, y \rangle$ for each $y \in I_x$.

$$T = \{(a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (b \leftrightarrow \neg b'), (b' \leftrightarrow \neg b), (y_1 \leftrightarrow (\neg a \wedge \neg b \wedge \neg y_1))\} \tag{1.16}$$

Using $\mathcal{G}$ on the above GNF theory gives us the following graph:

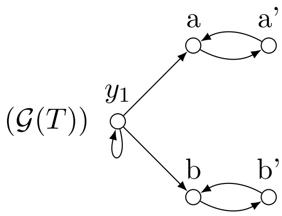

Figure 1.2

Again, will we not be proving the correspondence, but notice that T has three solutions, where either a, b or both are assigned 1. This reflects onto the graph where y_1 has to be assigned 0, thus forcing a or b to be assigned 1. The fact that $\mathcal{G}$ gives us the correspondence we are looking for is shown in [3].

With the problem of solutions in the graph being equivalent with SAT, we get our final equivalence between kernels in the graph and models of the theory. More precisely, we have that the set of kernels in a graph is equivalent with the set of models of the corresponding theory. Because of this tight link, we sometimes refer to graphs without kernels as paradoxical graphs.

The applicability of kernels should by now be obvious. In the next section we will review various findings within kernel theory and especially the findings related to infinitary graphs.

1.5 Results in kernel theory

The end goal within kernel theory is ultimately to develop an easy way of answering the question "Does this digraph have a kernel?" no matter the graph, and no matter the answer. As of today, we are not quite there, but a lot of work has been put into trying to identify special circumstances under which one is guaranteed to have (or guaranteed to not have) a kernel in the given graph. One significant result is the theorem proven by Moses Richardson in 1953:

Theorem 6 ([7])**.** *If D is a finitary digraph without odd cycles, then D has a kernel.*

Intuitively, one might be tempted to believe that *all* digraphs without odd cycles have kernels, but this is not the case. Until now, all our paradoxes have been statements that – directly or indirectly – have been referring back to themselves (giving cycles in the graph) and thus causing a logical conflict, and it is hard to imagine any other way to construct paradoxical statements. The following construction will however reveal our lack of imagination.

The *Yablo Graph*[8] is an example of an acyclic graph with no kernel. It is constructed with an infinite set of vertices $\{x_i \mid i \in \mathbb{N}\}$ and a set of edges N such that $\langle x_i, x_j \rangle \in N$ iff $i < j$.

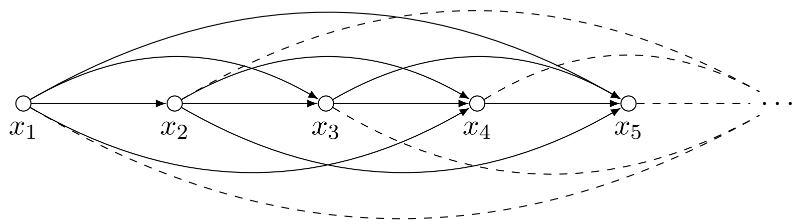

Figure 1.3: The Yablo Graph

Since the <-relation is a strict ordering, we have that the Yablo-graph indeed is acyclic. Furthermore, since any natural number has infinitely many numbers strictly larger than it, we get that all the vertices are infinitely branching, making the Yablo-graph infinitary.

The discourse represented by the Yablo-graph would – informally – be the situation with an infinite number of statements, all saying "Every statement after this statement is false".

We will later show formally that the Yablo-graph is indeed without a kernel, but for now the following explanation should suffice:

Suppose that the Yablo-graph *has* a kernel and that the vertex x_a is in it. Then all the vertices to the right of x_a are necessarily outside of the kernel, including x_{a+1}. But if x_{a+1} is outside of the kernel, it has to point to a vertex on the inside. This is now impossible, since the out-neighborhood of x_{a+1} is a subset of the out-neighborhood of x_a. Since x_a was chosen without any restrictions, no vertex can be inside the kernel, making it empty. Since no kernel can be empty, we have a contradiction, making the Yablo-graph without a kernel.

One thing should be mentioned at this point: The inverse of Richardson's statement is not valid; neither odd cycles nor infinitely branching vertices *entail* that their respective graphs are paradoxical. The following two graphs illustrate this point:

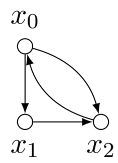

Figure 1.4

The above graph contains an odd cycle, but the singleton set $\{x_2\}$ is a kernel.

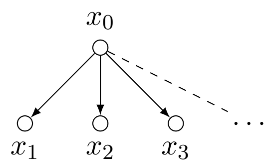

Figure 1.5

The above graph has an infinitely branching vertex x_0, but the infinite set $\{x_i \mid x > 0\}$ is a kernel.

Another important theorem within kernel theory is shown by Roy T. Cook in [6], stating that every digraph with at least one edge can be transformed into an infinitary dag – preserving and reflecting the solutions. This means that for any finitary graph that is paradoxical by the virtue of having an odd cycle, there is a corresponding infinitary, *acyclic* digraph that is also paradoxical. So if one is trying to find ways to identify paradoxical graphs, one does only need to look at dags.

1.6 Recognizing dags without kernels

Knowing that any graph can be translated to an equisatisfiable dag, the challenge is now to find sufficient conditions for dags to have kernels, even weaker than the one proved by Richardson (the fact that any finitary dag has a kernel is a direct consequence of Richardson's Theorem).

Definition 7. Given a graph $\mathbf{G} = \langle G, N \rangle$, a *ray* is an semi-infinite path, i.e. an infinite sequence $(x_1, x_2, \dots)$ of distinct vertices of G such that $(x_i, x_{i+1}) \in N$ for each i.

Definition 8. A vertex x_0 *dominates* a set of vertices $Y \subseteq G$ if there exists an infinite number of disjoint paths from x_0 to distinct vertices of Y.

Michał Walicki has proposed the following conjecture:

Conjecture 9. *If a dag has no kernel then it has a ray with infinitely many vertices dominating it.*

The contrapositive of Walicki's thesis suggests a condition for a kernel. This condition is weaker than the one from Richardson's Theorem, since a dag having a ray with infinitely many vertices dominating it implies that the dag is infinitary.

1.7 Resolving GNF-theories

In this section, I present an inference system introduced by Walicki in [9] which reasons over clausal theories induced from GNF-theories. The system is refutationally complete as well as non-explosive, allowing us to identify consistent parts of paradoxical discourse theories.

We later show some correlations between certain graph structures and clauses provable in this inference system.

Recall that a GNF theory consists exclusively of formulae of the following form:

$$x \leftrightarrow \bigwedge_{y \in I_x} \neg y \tag{1.17}$$

Using simple operations only, these formulae can be translated into an equivalent set of clauses. We start by writing the above bi-implication as two implications:

$$x \to \bigwedge_{y \in I_x} \neg y \qquad \text{and} \qquad x \leftarrow \bigwedge_{y \in I_x} \neg y \tag{1.18}$$

The first implication can be rewritten in the following way:

$$x \to \bigwedge_{y \in I_x} \neg y \quad \Leftrightarrow \quad \neg x \vee \bigwedge_{y \in I_x} \neg y \quad \Leftrightarrow \quad \bigwedge_{y \in I_x} (\neg x \vee \neg y) \quad \Leftrightarrow \quad \bigwedge_{y \in I_x} \neg(x \wedge y) \tag{1.19}$$

The second implication can be rewritten in the following way:

$$x \leftarrow \bigwedge_{y \in I_x} \neg y \quad \Leftrightarrow \quad x \vee \neg \left(\bigwedge_{y \in I_x} \neg y \right) \quad \Leftrightarrow \quad x \vee \bigvee_{y \in I_x} y \tag{1.20}$$

By splitting the conjunction from the first implication up into individual clauses, we get the following two kinds of clauses for every variable x in the GNF theory:

$$\text{OR-clause:} \quad x \vee \bigvee_{y \in I_x} y \tag{1.21}$$

$$\text{NAND-clauses:} \quad \neg(x \wedge y), \text{ for every } y \in I_x \tag{1.22}$$

We will treat both the OR-clauses and the NAND-clauses as sets of atoms, denoting NAND-clauses $\neg(x \wedge y)$ as $\overline{xy}$ and OR-clauses $x \vee y_1 \vee y_2 \vee y_3$ as $xy_1y_2y_3$. This enables us to state things like $\overline{xy} \subset \overline{xyz}$. A theory will – as expected – be a set of OR- and NAND-clauses.

If we interpret the initial GNF-theory as a graph $\mathbf{G} = \langle G, N \rangle$, for every vertex $x \in G$, there will be one OR-clause $\{x\} \cup N(x)$ and for every edge $\langle x, y \rangle \in N$ there will be a NAND-clause $\overline{xy}$.

The graphs from Figure 1.1 will have the following clausal theories:

$$\mathcal{T}(\mathbf{G_1}) = \{a, \overline{a}\} \tag{1.23}$$

$$\mathcal{T}(\mathbf{G_2}) = \{abc, b, c, \overline{ab}, \overline{ac}\} \tag{1.24}$$

$$\mathcal{T}(\mathbf{G_3}) = \{abc, bc, \overline{ab}, \overline{bc}\} \tag{1.25}$$

Further notation: $A \subseteq G$ denotes an OR-clause while $\overline{A} \subseteq G$ denotes a NAND-clause. Given a graph $\mathbf{G} = \langle G, N \rangle$, we denote the set of all NAND-clauses induced from the graph as NAND and all induced OR-clauses as OR. The combined set $\Gamma = NAND + OR$ will be our initial clauses in the inference system.

1.7.1 The inference system

We consider the following inference system, but we will focus on proofs using the axioms together with the (Rneg)-rule.

$$\text{(Ax)} \quad \Gamma \vdash C, \quad \text{for } C \in \Gamma \tag{1.26}$$

$$\text{(Rneg)} \quad \frac{\{\Gamma \vdash \overline{a_i A_i} \mid i \in I\} \quad \Gamma \vdash \{a_i \mid i \in I\}}{\Gamma \vdash \overline{\bigcup_{i \in I} A_i}} \tag{1.27}$$

$$\text{(Rpos)} \quad \frac{\Gamma \vdash A \quad \{\Gamma \vdash B_i K_i \mid i \in I\} \quad \{\Gamma \vdash \overline{a_i k} \mid i \in I, k \in K_i\}}{\Gamma \vdash (A \setminus \{a_i \mid i \in I\}) \cup \bigcup_{i \in I} B_i} \tag{1.28}$$

(Rneg) is creating NAND-clauses from NAND-clauses using OR as a side-condition. (Rpos) is creating OR-clauses from OR-clauses using NAND as a side-condition. In (Rneg), $\overline{a_i A_i}$ denotes the NAND $\overline{\{a_i\} \cup A_i}$ with a potentially empty A_i.

A proof in this system is a well-founded derivation, i.e. each of the branches in the proof tree is finite. This allows us to induce over the complexity of the proof tree.

The premise of the (Rneg)-rule is a set of I NAND-clauses together with one OR-clause with I elements such that each atom a_i in the OR-clause is contained within a NAND-clause, and such that each NAND-clause contains an atom from the OR-clause. The correspondence between the NAND-clauses and the elements of the OR-clause should in other words be bijective. The conclusion is the union of all the NAND-clauses without their corresponding atom from the OR-clause.

Whenever it is obvious and/or irrelevant from what theory we are proving something, we leave out this information in the proof to ease readability. Additionally, we move the single OR-clause in the premise to the side of the proof to emphasize its role as a side condition. These conventions are illustrated below:

$$\frac{\Gamma \vdash \overline{ax} \qquad \Gamma \vdash \overline{by} \qquad \Gamma \vdash ab}{\Gamma \vdash \overline{xy}} \quad \sim \quad \frac{\overline{ax} \qquad \overline{by}}{\overline{xy}}\, ab$$

Figure 1.6

Here are some examples of incorrect applications of the (Rneg)-rules, followed by some correct applications:

$$(1)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{cz}}{\overline{xyz}} abx \qquad (3)\ \frac{\overline{ax} \quad \overline{by}}{\overline{xy}} abx \qquad (2)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{bz}}{\overline{xyz}} abx$$

(1) is incorrect because the NAND $\overline{cz}$ contains no atoms from the OR abx. (2) is incorrect because the number of NAND-clauses does not match the length of the OR-clause. (3) is incorrect because there exist no bijective correspondence of the type described above.

$$(4)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{cz}}{\overline{xyz}} abc \qquad (5)\ \frac{\overline{ax} \quad \overline{b}}{\overline{x}} ab \qquad (6)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{xyz}}{\overline{xyz}} abx$$

The above applications are all correct, since all the atoms in each OR-clause get matched to exactly one NAND-clause in such a way that no NAND-clause stays unmatched.

The proof system sets no restrictions on the number and cardinality of its clauses, meaning that there might be an infinite number of clauses, and both the OR-clauses and the NAND-clauses might be either finite or infinite in size. Note that an infinite graph gives infinitely many NAND- and OR-clauses, while an infinitary graph also gives us infinitely *long* OR-clauses.

We study the refutation system that arises from the axioms and the (Rneg)-rule, calling it Neg. It is shown in [9] that Neg is sound for arbitrary theories and refutationally complete for theories with a countable number of OR-clauses. Soundness gives us for any graph $\mathbf{G} = \langle G, N \rangle$ that proving $\overline{C}$ for any $C \subseteq G$ implies that the vertices in C cannot all be assigned 1 in the graph model $\mathcal{T}(\mathbf{G})$. Refutational completeness gives us the property that whenever a graph/theory is inconsistent, we are able to prove $\varnothing$ in Neg.

We use the notation $\Gamma \vDash \overline{C}$ to express that every model of the theory Γ satisfies the NAND-clause $\overline{C}$. Note that because of the model/kernel equivalence presented in Section 1.4, the notation can also be used to express that none of the kernels in the graph $\mathcal{G}(\Gamma)$ contains all the vertices in the set C.

Since refutational completeness is only proven for theories with countably many OR-clauses, this thesis will only consider graphs with a countable number of vertices. We are thus able to assume both soundness and refutational completeness for all following graph theories.

Note that Neg is not generally complete, i.e. we do not always have that $G \vDash x \;\Rightarrow\; G \vdash \overline{x}$. The inconsistent graph G in Figure 1.7 exemplifies this:

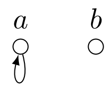

Figure 1.7

Because of the loop on vertex a, the graph has no solutions. We therefore have $G \vDash \varnothing$ and thus also $G \vDash \overline{b}$, but we are unable to prove $\overline{b}$ in Neg.

1.7.2 Inconsistency of the Yablo-graph

The inconsistency of the Yablo-graph is easily proven using Neg only. Since every vertex x_i (using the notation from Figure 1.3) has an edge to each vertex x_j where $j > i$, we get that every pair of distinct vertices is connected by an edge. This means that our set of axioms from the Yablo-graph looks like this:

$$\text{NAND} = \{\overline{x_i x_j} \mid i < j\} \qquad \text{OR} = \{x_i x_{i+1} x_{i+2} \ldots \mid i \in \mathbb{N}\} \tag{1.29}$$

For any vertex x_i from the Yablo-graph, we are now able to prove $\overline{x_i}$ in the following way:

$$\frac{\overline{x_i x_{i+1}} \quad \overline{x_i x_{i+2}} \quad \overline{x_i x_{i+3}} \quad \ldots}{\overline{x_i}}\; x_i x_{i+1} x_{i+2} \ldots$$

Figure 1.8

Proving $\varnothing$ is now simple:

$$\frac{\dfrac{\ldots}{\overline{x_1}} \quad \dfrac{\ldots}{\overline{x_2}} \quad \dfrac{\ldots}{\overline{x_3}} \quad \ldots}{\varnothing}\; x_1 x_2 x_3 \ldots$$

Figure 1.9

A less trivial inconsistency proof is the one of the *Stretched Yablo-graph*. This proof can be found in Appendix A.2 together with the definition of Stretched Yablo.

It is worth mentioning that even though our focus has been – and will be – on theories originating from graphs, the results on soundness and completeness hold for any theory consisting of a set of NANDs and a set of ORs.

An example of this is the pigeonhole problem which easily can be represented as a set of NAND- and OR-clauses, but does not directly correspond to a graph (it can of course be translated to a graph theory, like any other propositional theory). A Neg-proof of the pigeonhole principle can be found in Section **??**.

1.8 Thesis overview

The soundness and refutational completeness of Neg makes it a potentially great tool in the overarching endeavor of weakening the conditions for kernels in graphs. We know that for any graph G, $\varnothing$ can be proven in Neg based on G if and only if G is without a kernel.

Suppose that any inconsistency can be proven using only NAND-clauses of length at most 2, and that all NAND-clauses of length 2 correspond to vertices related in a certain way in the graph. This would in combination give us a condition for kernels, namely the absence of such specifically related vertices. With this in mind, the thesis will set out to do two things:

(1) Define a graph structural relation such that two vertices a, b in a graph G are so related if and only if $G \vdash \overline{ab}$.

This will be covered in Chapter 2 where we will be looking at increasingly general graph structures that ensure the provability of certain NAND-clauses, ultimately attempting to reach a structure general enough so that the provability of a NAND-clause entails the existence of that structure.

(2) Explore whether Neg is still refutationally complete when restricted to using NAND-clauses of length 1 or 2 only.

It will be shown in Chapter 3 that this is *not* possible. The chapter will present graphs where certain NAND-clauses are provable in general, but not under the restrictions mentioned. This will then be used to show that Neg, when under the restrictions of only using NAND-clauses of length 1 or 2, is unable to prove inconsistencies in certain graphs.

Chapter 2

NAND-clauses in graphs

Definition 10. A clause is *unary* if it contains only one atom. A clause is *binary* if it consists of one or two atoms.

In this chapter, we will motivate and conduct the search for graph structural equivalents of binary NAND-clauses provable in Neg. An actual graph structure corresponding to the binary NAND-clauses has not been found, but we will show various graph structures implying the provability of certain binary NAND-clauses.

2.1 Motivation

Since our proof system, Neg, has only one rule, the last step of any inconsistency proof will always look the same:

$$\frac{\overline{k_1} \quad \overline{k_2} \quad \overline{k_3} \quad \ldots}{\varnothing} \; k_1 k_2 k_3 \ldots$$

Figure 2.1

The premise will always consist of a collection of unary NAND-clauses, together with an OR-clause equal to the union of all the NAND-clauses. It is easy to see that none of the NAND-clauses can be larger than unary, since that would result in a non-empty NAND-clause in the conclusion. The OR-clause has to equal the union of the NAND-clauses simply by definition of the (RNeg)-rule.

This fact was also observed in [9]:

$$\Gamma \vdash \{\} \Leftrightarrow \exists K \in \mathrm{OR} : (\forall k \in K : \Gamma \vdash \overline{k}) \tag{2.1}$$

We know from the definition that any OR-clause used in the proof system corresponds to a single vertex with its successors in the graph. We do however not know what the

NAND-clauses of length 1 might correspond to. Knowing this would, by soundness and completeness of Neg, give us a graph structural condition for a kernel not to exist.

The only thing we *do* know about unary NAND-clauses is that they correspond to vertices that are assigned 0 in all solutions of the graph. We get this from soundness of Neg. As an example, consider the axiomatic unary NAND-clauses, which corresponds to loops in the graph. Vertices with loops can not be assigned 1 in any solution.

This is however not the graph structural property we are ultimately looking for, but at least we have reduced the question "What does an inconsistent graph look like?" to the question "What does a provably false vertex look like?".

So what does a unary NAND-clause proven in Neg correspond to in the graph? Similarly to a proof of $\varnothing$, there is really just one way to prove a unary NAND-clause:

$$\frac{\overline{xk_1} \quad \overline{xk_2} \quad \ldots \quad \overline{k_n} \quad \overline{k_{n+1}} \quad \ldots}{\overline{x}}\; k_1 k_2 \ldots k_n y_{k+1} \ldots$$

Figure 2.2

Any derivation of a unary NAND-clause $\overline{x}$ must end with a rule application using $K \in$ OR where for each $k \in K$, there is a NAND-clause in the premise that is either unary, $\overline{k}$ or binary, $\overline{xk}$. We require the premise to contain at least one binary NAND-clause, in order to actually be able to conclude with $\overline{x}$ and not $\varnothing$.

In other words:

$$\Gamma \vdash \overline{x} \Leftrightarrow \exists K \in \mathrm{OR} : \begin{pmatrix} \exists k \in K : G \vdash \overline{kx} \wedge \\ \forall k \in K : G \vdash \overline{kx} \vee G \vdash \overline{k} \end{pmatrix} \tag{2.2}$$

Just as we reduced the problem of inconsistency to the problem of unary NAND-clauses, we are able to reduce the problem further to binary NAND-clauses. We could even continue the reduction further to ternary clauses, quaternary clauses and so on, but without a change of strategy at some point, this seems pointless.[1]

Our current reduction lets us ask how two vertices x, y are *connected* in the graph when their binary NAND $\overline{xy}$ is proven in Neg. Observe that we have parts of this correspondence down already, with our NAND-axioms being all binary. Vertices directly connected by an edge must therefore be a part of the corresponding graph structure.

Let $R(a, b)$ denote the existence of some graph structure R between the vertices a and b in some graph G; our desired correspondence can now be formally defined as follows:

$$R(a, b) \quad \Rightarrow \quad G \vdash \overline{ab} \tag{2.3}$$

$$R(a, b) \quad \Leftarrow \quad G \vdash \overline{ab} \tag{2.4}$$

[1] We put a lot of attention on binary NAND-clauses because we at this point strongly believed that any inconsistency was provable using binary NAND-clauses only. This claim is investigated and disproved in the next chapter.

The two implications will be referred to as *implication (1)* and *implication (2)*, respectively. Note that since $\overline{a} = \overline{aa}$, we have the special case where $R(a, a) \Leftrightarrow G \vdash \overline{a}$.

Suppose we manage to find a structure R that satisfies both above implications. The following equation tells us how that would directly give us a predicate deciding whether or not the graph has a kernel.

$$
\begin{aligned}
&Sol(G) = \varnothing && (2.5)\\
\Leftrightarrow\ &G \vdash \varnothing && (2.6)\\
\overset{2.1}{\Leftrightarrow}\ &\exists K \in \mathrm{OR} : (\forall k \in K : G \vdash \overline{k}) && (2.7)\\
\overset{2.2}{\Leftrightarrow}\ &\exists K \in \mathrm{OR} : (\forall k \in K : (\exists L \in \mathrm{OR} : \begin{pmatrix} \exists l \in L : G \vdash \overline{lk} \wedge \\ \forall l \in L : G \vdash \overline{lk} \vee G \vdash \overline{l} \end{pmatrix})) && (2.8)\\
\underset{2.4}{\overset{2.3}{\Leftrightarrow}}\ &\exists K \in \mathrm{OR} : (\forall k \in K : (\exists L \in \mathrm{OR} : \begin{pmatrix} \exists l \in L : R(l, k) \wedge \\ \forall l \in L : R(l, k) \vee R(l, l) \end{pmatrix})) && (2.9)
\end{aligned}
$$

The following sections will present various graph structures, each satisfying implication (1). Each presented structure will be a generalization of the previous one, ultimately aiming to find a structure general enough to satisfy both implication (1) and implication (2). Such a structure would be a graph structural equivalent of a binary NAND provable in Neg. Unfortunately, we did not manage to find such a graph structure in this thesis.

For each structure presented, implication (1) will be shown, followed by an example disproving implication (2). The counterexamples will be graphs with the presented structure absent, but with the corresponding NAND-clause still provable.

Because of soundness of Neg, if two vertices correspond to a provable binary NAND in Neg, for any kernel K in that graph, at least one of the two vertices is outside K. The contrapositive of this observation being that for a graph G, if there exists a kernel $K \subseteq G$ such that two vertices, x_1 and x_2 are both in K, then $\overline{x_1 x_2}$ is not provable from G. This fact will be used when arguing why some graph structures are *not* satisfying implication (1).

2.2 Odd paths

We already know that whenever two vertices are connected by an edge, their binary NAND is trivially provable in Neg, since it is a part of the axioms.

We illustrate this case in Figure 2.3, where dashed lines represent possible out-edges to irrelevant parts of the graph. If a vertex has no dashed edges, it means that we disallow any additional edges out from this vertex.

This figure is the most basic example of a structure between two vertices that satisfies implication (1).

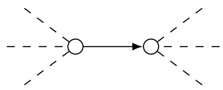

Figure 2.3

It is however easy to find an example showing how implication (2) does not hold.

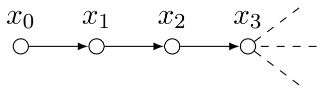

Figure 2.4

The above graph has the axioms NAND $= \{\overline{x_0x_1}, \overline{x_1x_2}, \overline{x_2x_3}\}$ and OR $= \{x_0x_1, x_1x_2, x_2x_3\}$. From these axioms, we can now, despite the fact that the vertices x_0 and x_3 are not connected by an edge, easily prove the NAND-clause $\overline{x_0x_3}$:

$$x_1x_2 \; \frac{\overline{x_0x_1} \qquad \overline{x_2x_3}}{\overline{x_0x_3}}$$

Figure 2.5

Intuitively, one can imagine that the proof above is connecting two NAND-clauses using an OR-clause, resulting in a new binary NAND-clause containing vertices that are weaklier connected than the ones we started with. This can be done repeatedly, resulting in the ability to prove binary NAND-clauses from vertices that are connected by arbitrarily long odd paths of the kind above.

Consider the following graph:

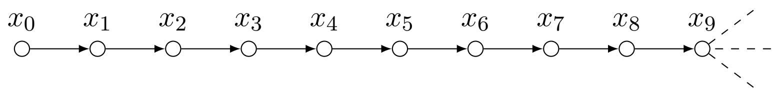

Figure 2.6

With axioms from the above graph, $\overline{x_0x_9}$ can be proven in Neg in the following way:

$$x_7x_8\,\dfrac{x_5x_6\,\dfrac{x_3x_4\,\dfrac{x_1x_2\,\dfrac{\overline{x_0x_1}\quad\overline{x_2x_3}}{\overline{x_0x_3}}\quad\overline{x_4x_5}}{\overline{x_0x_5}}\quad\overline{x_6x_7}}{\overline{x_0x_7}}\quad\overline{x_8x_9}}{\overline{x_0x_9}}$$

Figure 2.7

Observe that the above proof also proves $\overline{x_0x_3}$, $\overline{x_0x_5}$ and $\overline{x_0x_7}$ along the way, all of which contain vertices connected by paths of *odd* length. This is an important point. NAND-clauses containing vertices connected only by paths of *even* length cannot be proven in the same manner as above.

In many cases, such NAND-clauses cannot be proven at all. This is exemplified by the below graph, with a kernel containing the vertices a and b, showing that the NAND-clause $\overline{ab}$ is unprovable in Neg. The kernel in the graph below is represented by the black vertices.

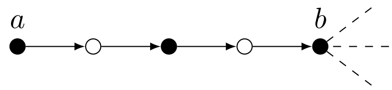

Figure 2.8

Restricting our paths to be of odd length lets us avoid cases like the one above. It is however not the case that all odd paths satisfy implication (1).

2.3 Trimming

We introduce the following terminology:

Definition 11. Given a path and two consecutive vertices x and y from that path, we will say that x is *trimmed*, with respect to that path, if $N(x) = \{y\}$.

Note that when a vertex is trimmed, its corresponding OR-clause is binary.

Definition 12. If all the vertices of a path, except the terminal vertex, are trimmed, we will call the path a *fully trimmed path.*

All the paths presented in this chapter so far have been fully trimmed.

If two vertices a and b are connected by an odd path that is *not* fully trimmed, $\overline{ab}$ is not necessarily provable in Neg. The below graph exemplifies this with a kernel containing both a and b, even though they are connected by an odd path.

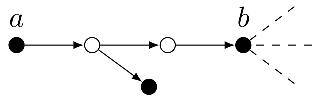

Figure 2.9

Whenever two vertices, x_0 and x_k, are connected by a fully trimmed path of odd length, $\overline{x_0x_k}$ is generally provable in Neg in the following way:

$$\dfrac{x_3x_4 \qquad \dfrac{x_1x_2 \qquad \overline{x_0x_1} \qquad \overline{x_2x_3}}{\overline{x_0x_3}} \qquad \overline{x_4x_5}}{\overline{x_0x_5}}$$

$$\vdots$$

$$\dfrac{x_{k-2}x_{k-1} \qquad \overline{x_0x_{k-2}} \qquad \overline{x_{k-1}x_k}}{\overline{x_0x_k}}$$

Figure 2.10

Since the path is fully trimmed, all the OR-clauses used are binary, letting us prove our NAND-clause without introducing any other clauses than the ones we get from the path itself. However, notice that only half of the OR-clauses is actually in use in such a proof. Thus, we need only to restrict half of the vertices in the path to not branch.

With the proof from Figure 2.10 in mind, consider the following graph:

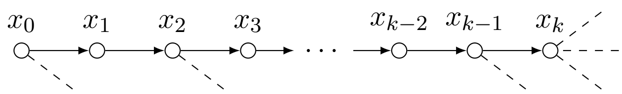

Figure 2.11: An oddly trimmed path of odd length

In the above path, between x_1 and x_k, only every other vertex is trimmed. We will call this path variant an *oddly trimmed path*, and define it formally as follows:

Definition 13. A path $(x_0, x_1, \ldots, x_k)$ is *oddly trimmed* if for each odd $i < k$, x_i is trimmed with respect to that path.

One can immediately note that any fully trimmed path is also oddly trimmed.

The axioms we get from the oddly trimmed path above do not differ significantly from the fully trimmed variant. Since the vertices $x_0, x_2, x_4, \ldots, x_{k-1}$ no longer have single successors, their corresponding OR-clauses will no longer be binary. However, since none of these OR-clauses are used in the general proof in Figure 2.11, the proof will remain valid also for the oddly trimmed path.

This makes us able to generalize further and say that any two vertices connected by an oddly trimmed path of odd length satisfy implication (1).

2.4 Odd vels

The following observation will further generalize the concept of oddly trimmed paths:

The direction of edges in a graph can often be changed individually while still keeping many axiomatic clauses unchanged, including all the NAND-clauses.

Consider the following graph:

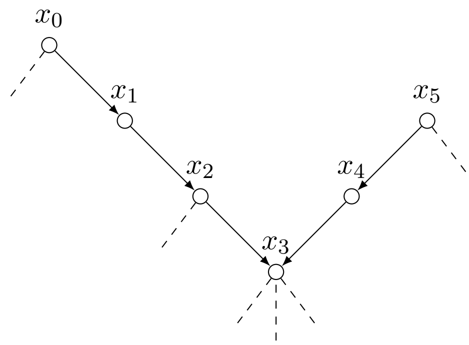

Figure 2.12

From the above graph, $\overline{x_0 x_5}$ can be proven in the same way as in the proof of $\overline{x_0 x_9}$ in Figure 2.7. Again, the only thing we are changing in terms of the axioms are the OR-clauses that are not used in the proof.

We will call this kind of graph structure an *odd vel* and define it formally in the following way:

Definition 14. Two vertices a and b have an *odd vel* between them if there exists a vertex c such that there are oddly trimmed paths from a to c and from b to c, one of even length (possibly 0) and one of odd length.

Notice that an oddly trimmed path of odd length is just an instance of an odd vel where the even path is of length 0.

The formal proof why all NAND-clauses from vertices connected by such vels are provable in Neg can be found in Appendix A.3.

There are obviously other ways of altering directions of individual edges in a path. The next example will however show that most of them will alter the OR-clauses in such a way that implication (1) no longer hold.

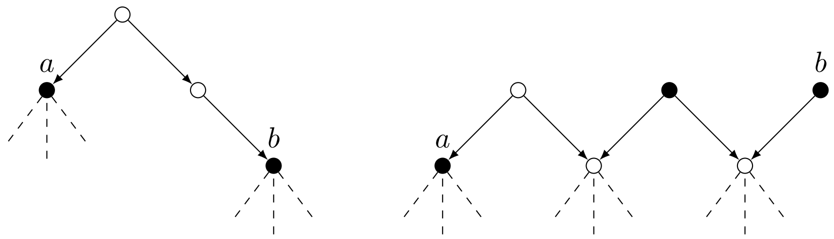

Figure 2.13

The figure above shows two odd paths that have had some of their edges flipped. In both examples, the shown kernels contain both a and b, showing that neither gives a provable NAND-clause $\overline{ab}$ in Neg.

2.5 Generalizing trimming

In this section, we will take a closer look at the current concept of trimming, and through some examples introduce a more general definition. So far, trimming has meant forcing certain vertices to point only at its successor in a path.

In Section 2.2 we motivated the concept of trimmed paths by presenting the graph in Figure 2.9. The figure showed how adding out-edges to certain vertices in the odd path makes the corresponding NAND-clause unprovable.

Notice however that adding a loop on the vertex to which the path branches, leaves us with a different situation.

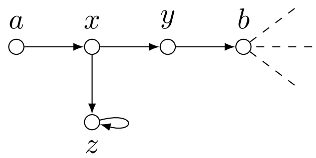

Figure 2.14

The addition of the loop adds $\overline{z}$ to the axioms. The NAND-clause $\overline{ab}$ can now easily be proven in the following way:

$$xyz\ \frac{\overline{ax} \quad \overline{yb} \quad \overline{z}}{\overline{ab}}$$

Figure 2.15

This shows the fact that vertices at odd positions in a path can indeed branch off to other vertices than their path successor and still contribute to a provable NAND-clause, just as long as the vertices they branch off to satisfy certain criteria. If the vertex is provably false, we can use that unary NAND-clause, like we did in the above proof, to handle the non-binary OR-clauses. This observation lets us generalize our definition of trimmed vertices and still have our vel-relation satisfy implication (1).

The new, general notion of trimming can be formally defined like follows:

Definition 15. Given a path in some graph $\mathbf{G} = \langle G, N \rangle$ and two consecutive vertices x and y from that path, we say that x is *trimmed*, with respect to that path, if $\forall x \in G : (c \in N(x) \setminus \{y\} \Rightarrow \mathcal{T}(\mathbf{G}) \vdash \bar{c})$.

The problem with this generalized definition is that it is using the notion of provability, giving a vel-definition that is not purely graph-structural. With such a definition, whenever we want to check whether two vertices have a vel between them, we may have to work out the actual provability of certain unary NAND-clauses, which is exactly what we try to avoid with our vel-relation.

In addition to this problem, consider the following two graphs:

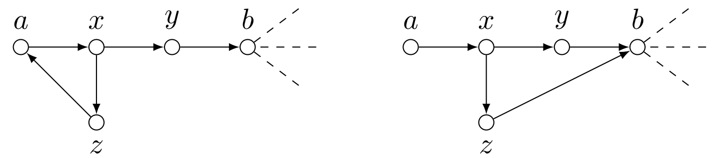

Figure 2.16

Like with the graph from Figure 2.14, the above graphs are both based on Figure 2.9, each with only one edge added. The left variant has $\overline{az}$ in its axioms, while the right variant has $\overline{bz}$, again making their respective proofs of $\overline{ab}$ almost trivial.

$$xyz\ \frac{\overline{ax} \quad \overline{yb} \quad \overline{za}}{\overline{ab}} \qquad\qquad xyz\ \frac{\overline{ax} \quad \overline{yb} \quad \overline{zb}}{\overline{ab}}$$

Figure 2.17

The above proofs do not rely on any provably false vertex, but rather make use of the fact that z is connected to one of the vertices contained in the NAND-clause that we are trying to prove.

In the case of Figure 2.16, $\overline{ab}$ will be provable as long as for all the successors z of x, either $\bar{z}$, $\overline{az}$ or $\overline{bz}$ are provable in Neg.

This means that our definition of trimming should be generalized even further, taking into account the observation made above. Such a generalization must add the case where, given a vel between a and b, trimmed vertices may branch off to a vertex c as long as either $\overline{ac}$ or $\overline{bc}$ are provable in Neg.

In order to keep our definitions strictly graph structural , we change our approach and present two inductive definitions satisfying implication (1).

2.6 Inductive definitions

All the graph structures presented in this chapter so far can easily be defined inductively. For instance, given some graph $\mathbf{G} = (G, N)$, let $V_1 \subseteq G \times G$ denote the set of all pairs of vertices related by our original vel-definition (Definition 14) where 'trimmed' meant strictly non-branching.

We can define V_1 inductively in the following way, with a and b being arbitrary vertices from G.

$$\begin{array}{rll} \textbf{(BC):} & N(a,b) \vee N(b,a) & \Rightarrow V_1(a,b) \\ \textbf{(IS):} & \exists c \in N(a) : (N(c) = \{d\} \wedge V_1(d,b)) & \Rightarrow V_1(a,b) \end{array} \tag{2.10}$$

The symmetry in the base case is what makes this a definition of a vel and not a path, while the trimmed-ness gets expressed by the restrictions we set on vertex c in the inductive step.

2.6.1 The V_2 relation – a generalization of V_1

Using induction, it is now easier to formally define the new vel-relation corresponding to the concept of trimming from Definition 15, without using concepts of provability. Let $V_2 \subseteq G \times G$ be the set of all pairs of vertices related by the new vel-definition. V_2 can be defined inductively in the following way, again with a and b as arbitrary vertices from G.

$$\begin{array}{rll} \textbf{(BC):} & N(a,b) \vee N(b,a) & \Rightarrow V_2(a,b) \\ \textbf{(IS):} & \exists c \in N(a) : \left(\begin{array}{l} \exists d \in N(c) : V_2(d,b) \quad \wedge \\ \forall d \in N(c) : V_2(d,b) \vee V_2(d,a) \vee V_2(d,d) \end{array} \right) & \Rightarrow V_2(a,b) \end{array} \tag{2.11}$$

Comparing the two inductive definitions, it is easy to see that $V_1(a,b) \Rightarrow V_2(a,b)$.

The fact that $V_2(a,b) \ \Rightarrow \vdash \overline{ab}$ can now be proven inductively, showing that V_2 satisfies implication (1):

Proof. Given a graph G with two vertices a and b such that $V_2(a,b)$:

Base case: If $N(a,b)$ or $N(b,a)$, then the NAND-clause $\overline{ab}$ is an axiom.

Inductive step: The existence of a vertex $c \in N(a)$ gives us the axiomatic NAND-clause $\overline{ac}$. Letting $D = N(c)$ we also have that the OR-clause cD is an axiom. This gives us the following incomplete proof:

$$\frac{\overline{ac} \quad \dots}{\overline{ab}}\ cD$$

Let D_b, D_a and D_d be subsets of D such that for any $d \in D$:

$$d \in D_b \Leftrightarrow V_2(d,b), \quad d \in D_a \Leftrightarrow V_2(d,a), \quad d \in D_d \Leftrightarrow V_2(d,d), \tag{2.12}$$

as illustrated in Figure 2.18

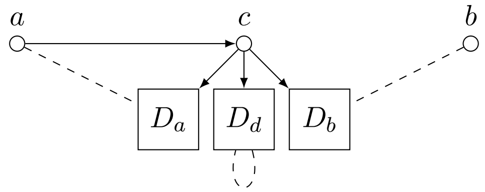

Figure 2.18: The inductive case of a V_2 construction. Boxes represents sets of vertices. An edge between a vertex and a box represents a set of edges from the vertex to each of the vertices in the set.

The inductive part of the V_2 definition (2.11) reveals two properties of the sets D_b, D_a and D_d:

- The set D_b is nonempty, since $\exists d \in N(c) : V_2(d,b)$.
- $D_b \cup D_a \cup D_d = D$, since $\forall d \in N(c) : V_2(d,b) \vee V_2(d,a) \vee V_2(d,d)$.

The induction hypothesis lets us assume the provability of the following sets of NAND-clauses: $\{\overline{db} \mid d \in D_b\}, \{\overline{da} \mid d \in D_a\}, \{\overline{d} \mid d \in D_d\}$. Inserting these clauses into the incomplete proof from Figure **??** gives the following proof:

$$\frac{\overline{ac} \qquad \{\overline{db} \mid d \in D_b\} \qquad \{\overline{da} \mid d \in D_a\} \qquad \{\overline{d} \mid d \in D_d\}}{\overline{ab}}\ cD$$

Figure 2.19

The nonemptiness of D_b gives us the existence of a b in the premise, while the fact that $D_b \cup D_a \cup D_d = D$ guarantees that each atom in the OR-clause D has a match in a NAND-clause.

The proof is thus valid, finishing the inductive step and ultimately proving the fact that $V_2(a,b) \Rightarrow \vdash \overline{ab}$. $\square$

Having that V_2 satisfies implication (1), if one could show that it also satisfies implication (2), the consequences would be considerable. It would mean that any provable binary NAND-clause would have a proof that in each step combines a set of already proved binary NAND-clauses with an axiom. Any provable NAND-clause could in other words be constructed by adding one fresh axiom at every step, hinting at a possible normal form of proofs in Neg.

Unfortunately, V_2 does *not* satisfy implication (2). The vertices a and b in the graph presented below will not be related by V_2, but $\overline{ab}$ is still provable in Neg. The graph will thus act as a counterexample for implication (2).

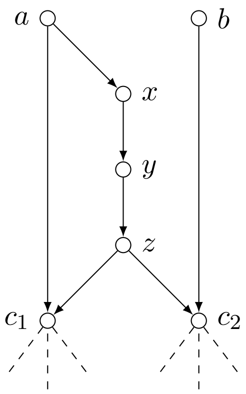

Figure 2.20

Given the above graph, $\overline{ab}$ can be proven like follows:

$$\dfrac{xy\,\dfrac{\overline{ax} \qquad \overline{yz}}{\overline{az}} \qquad \overline{ac_1} \qquad \overline{bc_2}}{\overline{ab}}\; zc_1c_2$$

Figure 2.21

We now show that the graph in Figure 2.20 is not an instance of V_2.

Suppose that $V_2(a, b)$. Since a and b are not connected by an edge, their relation has to be an instance of the inductive step. Because of the non-emptiness requirement, $\exists c \in N(a) : \exists d \in N(c) : V_2(d, b)$ from the V_2 definition, either a must have a successor c which again has a successor d such that $V_2(d, b)$ or b must have a successor c which again has a successor d such that $V_2(d, a)$.

The vertices a and b have a total of 3 successors, 2 of which only branch off to irrelevant vertices and therefore are unable to satisfy the non-emptiness requirement. The last successor is the vertex x, which only has one successor, vertex y.

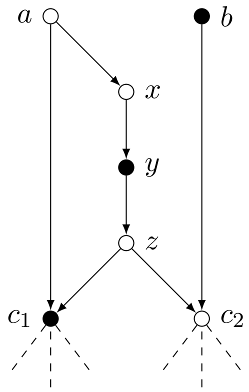

Figure 2.22

The above solution shows us by soundness of Neg that $\overline{yb}$ is unprovable, and therefore by implication (1) that y and b cannot be V_2-related. Since none of the successors of a and b satisfies the non-emptiness requirement, they also are not V_2 related. We can therefore conclude that V_2 does not satisfy implication (2).

2.6.2 The V_3 relation – a generalization of V_2

Now, in what way can V_2 be generalized in order to include the graph in Figure 2.20? Looking at the proof in Figure 2.19, we see that the axiomatic NAND-clause $\overline{ac}$ is a crucial part of the premise, representing the $c \in N(a)$ in the definition of V_2. Observe that a case where $\overline{ac}$ is non-axiomatic would not invalidate this rule application. In other words, the vertex c from the V_2 definition does not necessarily need to be in the neighborhood of a, they only need to be in a V_2-relation.

Based on this observation, we define the relation V_3, a generalization of V_2. Like in the definitions of V_1 and V_2, a and b are vertices from G.

$$\begin{array}{rll}
\textbf{(BC):} & N(a,b) \vee N(b,a) & \Rightarrow V_3(a,b) \\
\textbf{(IS):} & \exists c \in G : \left(\begin{array}{l} \exists d \in N(c) \cup \{c\} : V_3(d,a) \quad \wedge \\ \exists d \in N(c) \cup \{c\} : V_3(d,b) \quad \wedge \\ \forall d \in N(c) \cup \{c\} : V_3(d,a) \vee V_3(d,b) \vee V_3(d,d) \end{array} \right) & \Rightarrow V_3(a,b)
\end{array} \tag{2.13}$$

V_2 required a to have an edge to c. We are now treating c just like its successors by requiring it to be in V_3-relation to either a, b or itself. Requiring a to be in V_3-relation to some vertex in $N(c) \cup \{c\}$ is now what ensures the inclusion of a in the proof.

Looking back at the graph from Figure 2.20, we see that vertex z is clearly V_3-related to a while its successors, c_1 and c_2, are V_3-related to a and b, respectively. The vertices a and b are thus V_3-related.

Implication (1) for V_3 is proven in more or less the same way as we proved implication (1) for V_2 in Section 2.6.1. Therefore, only a condensed version will be shown here:

Proof. Given a graph G with two vertices a and b such that $V_3(a, b)$:

Base case: Same as in V_2 proof.

Inductive step: Let $D = N(c) \cup \{c\}$ and let D_a, D_b and D_d denote the same sets as in the V_2 proof. The definition of V_3 tells us that both D_a and D_b are nonempty. This fact together with the fact that $D = D_a \cup D_b \cup D_d$ lets us form the following proof:

$$\frac{\{\overline{da} \mid d \in D_a\} \qquad \{\overline{db} \mid d \in D_b\} \qquad \{\overline{d} \mid d \in D_d\}}{\overline{ab}}\ D$$

Figure 2.23

No assumption was made on the vertices a and b, so we can conclude that $V_3(a, b) \Rightarrow\ \vdash\ \overline{ab}$. □

We propose the following conditional property for V_3:

Lemma 16. *If V_3 satisfy implication (2), then given a graph $\mathbf{G} = \langle G, N \rangle$ and any two vertices $a, b \in G$; if $V_3(a, b)$ then $\overline{ab}$ is provable in Neg using binary NAND-clauses only.*

Proof. Since the vertices a, b from the latest proof are chosen arbitrarily, we have from Figure 2.23 that any two V_3-related vertices are provable in such a way that the premise of the last rule application contains binary NAND-clauses only. Implication (2) would give us that each NAND-clause in the premise contains V_3-related vertices. These would then also have proofs with binary NAND-clauses only in their last premise. By repeatedly applying the same reasoning through the whole proof, we end up with a proof consisting of binary NAND-clauses only. □

Lemma 17. *V_3 does not satisfy implication (2), i.e., $\Gamma \vdash \overline{ab} \not\Rightarrow V_3(a, b)$*

Proof. The graph in Figure 2.24 provides a provable NAND-clause $\overline{ab}$, but is not an instance of V_3.

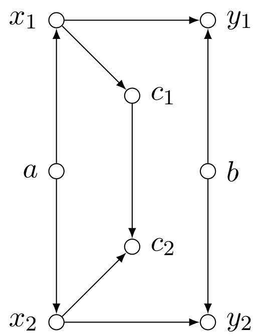

Figure 2.24

The graph is admittedly a bit over-simplified. Even though the vertices y_1, y_2 and c_2 are depicted like sinks, we treat them as initial vertices of (non-depicted) disjoint rays. We will however not consider the clauses corresponding to the elements of these rays, since these are not going to contribute to our argument.

Here is one way to prove $\overline{ab}$ in Neg given the graph theory from Figure 2.24:

$$c_1c_2\;\frac{x_2y_2c_2\;\dfrac{\overline{ax_2}\quad\overline{by_2}\quad\overline{c_1c_2}}{\overline{abc_1}}\qquad\qquad\dfrac{\overline{ax_1}\quad\overline{by_1}\quad\overline{c_1c_2}}{\overline{abc_2}}\;x_1y_1c_1}{\overline{ab}}$$

Figure 2.25

If implication (2) holds for V_3, we would expect a and b to be V_3-related in the above graph. In other words, we would expect there to be a third vertex in the graph such that it and each of its successors are V_3-related to either a, b or itself and such that at least one of them is V_3-related to a and one to b. This is not the case, as we now show.

Consider the following six solutions of the graph:

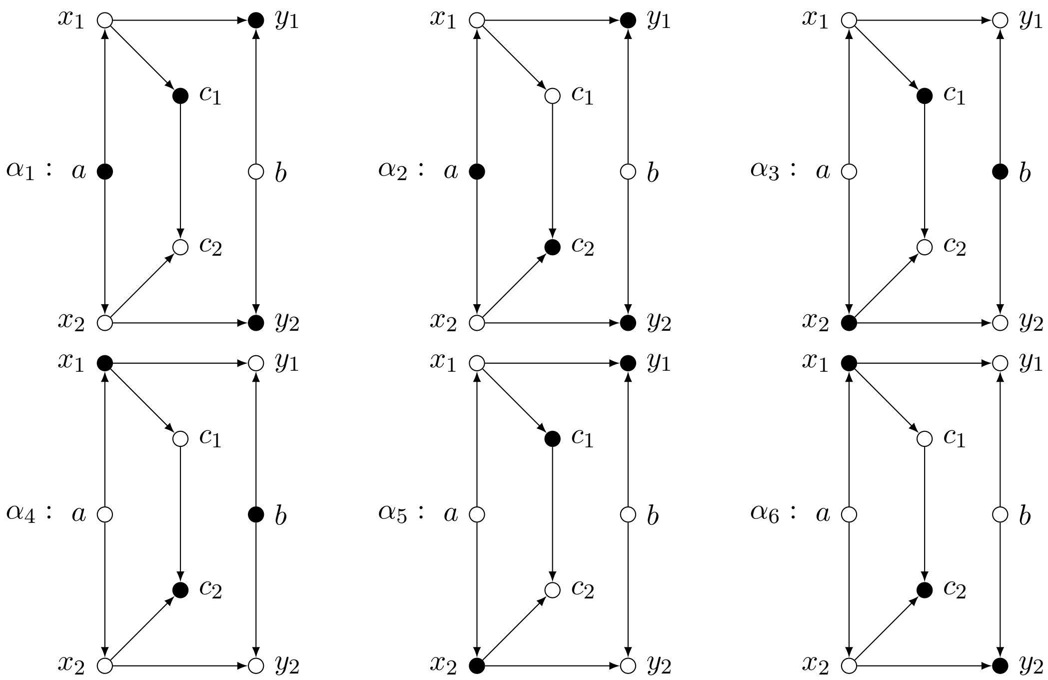

Figure 2.26

Given any pair of vertices x, y, if there exists a solution such that x and y are both assigned 1, then $\overline{xy}$ is not provable in Neg. We get this from soundness.

The table in Figure 2.27 shows what pairs of vertices can be 1 under the same solution and thus constitute a binary NAND-clauses unprovable in Neg. A cell is filled with a reference to the solution, if any, exemplifying the case. This relation is obviously symmetric, so we leave out the lower half of the table to ease readability.

	a	b	x_1	x_2	y_1	y_2	c_1	c_2
a	α_1				α_1	α_1	α_1	α_2
b	–	α_4	α_4	α_3			α_3	α_4
x_1	–	–	α_4			α_6		α_6
x_2	–	–	–	α_5	α_5		α_5	
y_1	–	–	–	–	α_1	α_1	α_1	α_2
y_2	–	–	–	–	–	α_1	α_1	α_2
c_1	–	–	–	–	–	–	α_1	
c_2	–	–	–	–	–	–	–	α_4

Figure 2.27

Two important observations are to be made from the above table:

First, for each vertex in the graph, there exists a solution where that vertex is assigned 1. By soundness, no unary NAND-clause can therefore be proven in Neg under this theory.

Secondly, among the pairs above that are not shown to be unprovable, only two are not axioms: $\overline{ab}$ and $\overline{x_1x_2}$. We are left with the following collection of 11 "not unprovable" binary NAND-clauses:

$$\{\ \overline{ab},\ \overline{ax_1},\ \overline{ax_2},\ \overline{by_1},\ \overline{by_2},\ \overline{x_1x_2},\ \overline{x_1y_1},\ \overline{x_1c_1},\ \overline{x_2y_2},\ \overline{x_2c_2}\ \overline{c_1c_2}\ \} \tag{2.14}$$

By the definition of V_3 from (**??**) and (**??**), in order for a and b to be V_3-related, a vertex s has to exist such that it and all its successors are V_3-related to either a, b or itself and such that at least one of them is V_3-related to a and one to b.

Looking at the table, one can see that most pairs containing a and b are unprovable and thus not V_3-related. Only $\overline{ax_1}$, $\overline{ax_2}$, $\overline{by_1}$ and $\overline{by_2}$ are not shown to be unprovable. Since no vertex is V_3-related to itself, the vertex s described above has to either be an x- or a y-vertex. None of the y-vertices have successors V_3-related to a or b. Both x-vertices have a c-vertex as a successor, none of which are V_3-related to a or b. We thus have that none of the vertices in the graph satisfy the requirements making a and b V_3-related. Since $\overline{ab}$ is provable in Neg, we get that $\Gamma \vdash \overline{ab} \not\Rightarrow V_3(a,b)$ so implication (2) fails. □

2.7 Concluding remarks

In the process of repeatedly generalizing our graph structural definitions, we have reached a situation where our current definition is almost identical to the actual axioms and rule-applications in our proof system; more specifically, the applications with binary NAND-clauses in their conclusion. This comes as no surprise, but is a bit disappointing. The original goal was to find some graph structural relation R, independent of the definition of Neg, such that for any graph G with vertices a, b: $R(a,b) \Leftrightarrow G \vdash \overline{ab}$ (implication (1) and (2)). This would in turn give us a graph structure present only when the graph in question was kernel free. It has however become apparent that no simple graph structural definition suffices in satisfying these implications. Even V_3 falls short in this endeavor.

The big lesson here might be that the existence of small substructures in graphs is not always sufficient in predicting the absence of kernels. Trying to recognize and isolate inconsistent parts of a graph seems to be the wrong approach in many cases. We will therefore at this point terminate our search for such a graph structure.

However, in the process of searching for these structures, we developed the counter-example disproving implication (2) of V_3 (Figure 2.24). This graph will in the next chapter be utilized to disprove the main hypothesis given for this thesis.

Chapter 3

Refutational incompleteness of BNeg

Definition 18. *BNeg* denotes the proof system Neg when restricted to using binary NAND-clauses only[1]. A clause is *binary-derivable* if it is provable in BNeg.

This chapter investigates the following conjecture:

Conjecture 19. *BNeg is refutationally complete for graph theories.*

This conjecture was given by supervisor as a main hypothesis for this thesis.

A proof for it would be significant both because it would be a strong property for a proof system in general, but also because it could potentially help us to further characterize a kernel-free graph. Having that any inconsistency can be proven in Neg using binary NAND-clauses only might imply a similar property in kernel-free graphs, namely that inconsistencies can be described as collections of pairwise structural relations between vertices.

Section 3.1 will disprove a variant of the conjecture, looking at general theories, not necessarily from graphs. Section 3.2 will prove that some binary NAND-clauses provable from graph theories are not binary-derivable. Section 3.3 will build on this result to show that there exist unary NAND-clauses provable from graph theories that are not binary-derivable. Lastly, Section 3.4 will present the final proof showing an inconsistency in a graph theory that is not binary-derivable, disproving Conjecture 19.

[1] Recall that *binary* also covers clauses of length 1

3.1 Inconsistencies in general theories

This section will show that there exist inconsistent theories such that their inconsistencies are not possible to prove in Neg using binary NAND-clauses only. We will prove the pigeonhole principle in Neg to exemplify this.

The pigeonhole principle states that whenever you have n pigeons and m holes such that $n > m$, then at least one hole must contain more than one pigeon. We can prove this principle in Neg by proving an inconsistency of the theory that (1) each of the n pigeons is contained in a hole and (2) each hole contains only one pigeon.

Letting the atom x_i denote the pigeon i occupying the hole x, the above theory, with n pigeons and m holes, can be formalized in the following way:

$$(1):\quad \{\ 1_i 2_i 3_i \dots m_i \mid i \leq n\ \} \tag{3.1}$$

$$(2):\quad \{\ \overline{x_i x_j} \mid i < j \leq n, x \leq m\ \} \tag{3.2}$$

We now prove the pigeonhole principle for 4 pigeons and 3 holes.

Proof. This instance of the pigeonhole principle has the following axioms:

$$\begin{gathered}
\mathrm{OR} = \{\ 1_1 2_1 3_1,\ 1_2 2_2 3_2,\ 1_3 2_3 3_3,\ 1_4 2_4 3_4\ \} \\
\mathrm{NAND} = \left\{ \begin{array}{l}
\overline{1_1 1_2},\ \overline{1_1 1_3},\ \overline{1_1 1_4},\ \overline{1_2 1_3},\ \overline{1_2 1_4},\ \overline{1_3 1_4}, \\
\overline{2_1 2_2},\ \overline{2_1 2_3},\ \overline{2_1 2_4},\ \overline{2_2 2_3},\ \overline{2_2 2_4},\ \overline{2_3 2_4}, \\
\overline{3_1 2_2},\ \overline{3_1 3_3},\ \overline{3_1 3_4},\ \overline{3_2 3_3},\ \overline{3_2 3_4},\ \overline{3_3 3_4}
\end{array} \right\}
\end{gathered} \tag{3.3}$$

From the axioms above we can prove inconsistency by first proving the three NAND-clauses $\overline{1_4}$, $\overline{2_4}$ and $\overline{3_4}$:

$$\dfrac{\overline{1_3 1_4} \qquad \dfrac{\overline{1_2 1_4} \quad \overline{2_2 2_3} \quad \dfrac{\overline{1_1 1_4} \quad \overline{2_1 2_3} \quad \overline{3_1 3_2}}{\overline{3_2 2_3 1_4}}\ 1_1 2_1 3_1}{\overline{2_3 1_4}}\ 1_2 2_2 3_2 \qquad \dfrac{\overline{1_2 1_4} \quad \dfrac{\overline{1_1 1_4} \quad \overline{2_1 2_2} \quad \overline{3_1 3_3}}{\overline{2_2 3_3 1_4}}\ 1_1 2_1 3_1 \quad \overline{3_2 3_3}}{\overline{3_3 1_4}}\ 1_2 2_2 3_2}{\overline{1_4}}\ 1_3 2_3 3_3$$

Figure 3.1

$$\dfrac{\dfrac{\overline{1_21_3} \quad \overline{2_22_4} \quad \dfrac{\overline{1_11_3} \quad \overline{2_12_4} \quad \overline{3_13_2}}{\overline{3_21_32_4}}\, 1_12_13_1}{\overline{1_32_4}}\, 1_22_23_2 \quad \overline{2_32_4} \quad 1_22_23_2\, \dfrac{1_12_13_1\, \dfrac{\overline{1_11_2} \quad \overline{2_12_4} \quad \overline{3_13_3}}{\overline{1_23_32_4}} \quad \overline{2_22_4} \quad \overline{3_23_3}}{\overline{3_32_4}}}{\overline{2_4}}\, 1_32_33_3$$

Figure 3.2

$$1_32_33_3\, \dfrac{1_22_23_2\, \dfrac{\overline{1_21_3} \quad 1_12_13_1\, \dfrac{\overline{1_11_3} \quad \overline{2_12_2} \quad \overline{3_13_4}}{\overline{2_11_33_4}} \quad \overline{3_23_4}}{\overline{1_33_4}} \quad 1_22_23_2\, \dfrac{1_12_13_1\, \dfrac{\overline{1_11_2} \quad \overline{2_12_3} \quad \overline{3_13_4}}{\overline{1_22_33_4}} \quad \overline{2_22_3} \quad \overline{3_23_4}}{\overline{2_33_4}} \quad \overline{3_33_4}}{\overline{3_4}}$$

Figure 3.3

With these three NAND-clauses proven, we can now prove $\varnothing$ in one step:

$$1_42_43_4\, \dfrac{\overline{1_4} \quad \overline{2_4} \quad \overline{3_4}}{\varnothing}$$

Figure 3.4

$\square$

Observe that NAND-clauses of length 3 appear several times in this proof. We will show that this is unavoidable in Neg given the axioms from 3.3.

Lemma 20. *The inconsistency from the axioms in 3.3 is not binary-derivable.*

Proof. As observed in Section 2.1, the only strategy in proving inconsistencies in Neg is to create new NAND-clauses until you have a set of unary NAND-clauses such that their union matches an OR-clause.

Now, what possible ways are there to create new NAND-clauses from our given pigeonhole axioms? The OR-clauses are what dictates the ways new NAND-clauses can be created, and since all OR-clauses are of length 3, we get that any premise must consist of exactly 3 NAND-clauses. In addition, since all OR-clauses contain exactly one atom from each hole, each of the three NAND-clauses must contain atoms from different holes.

Looking at the NAND-clauses in the axiom set, we see that none of them contain atoms from two different holes, so the three axiomatic NAND-clauses eligible in a premise are mutually disjoint (each contains atoms from a hole different from the other two). Since all three are binary, we have a total of 6 different atoms in the premise, and with the OR-clause shaving of 3 of these, our conclusion must contain 3 different atoms. Any NAND-clause derived directly from axioms must therefore be of length 3.

$$1_i2_i3_i \ \frac{\overline{1_i1_j} \qquad \overline{2_i2_k} \qquad \overline{3_i3_l}}{\overline{1_j2_k3_l}}$$

Figure 3.5

It is easy to see that this generalizes to any version of the pigeonhole principle. When you have n holes and $> n$ pigeons, the OR-clauses will be of length n, requiring n mutually disjoint NAND-clauses in the premise of the first proof step. This results in a NAND-clause of length n when derived directly from axioms.

This means not only that we are unable to keep the clause length at 2, but also that the size of the NAND-clauses increases with the size of the OR-clauses, which in this case coincides with the number of holes.

Since $\varnothing$ is not a part of the axioms and not directly derivable from axioms, its proof has to be of height > 2 and must therefore include a NAND-clause of size equal to the size of the OR-clauses. Therefore, with the formulation in (3.3), the inconsistency in the pigeonhole principle is not binary-derivable. □

3.2 Binary NAND-clauses in graph theories

We have just shown that in the case of unrestricted theories, there is no guarantee that an inconsistency is binary-derivable, but what about graph theories? After all, every theory can be represented as an equisatisfiable graph theory.

It turns out that even for graph theories, some provable NAND-clauses require non-binary NAND-clauses in their proof, i.e they are *not* binary-derivable. This section will disprove the following conjecture:

Conjecture 21. *Given a graph theory, any provable binary NAND-clause is binary-derivable.*

The conjecture will be disproved simply by presenting a graph containing a provable binary NAND-clause and show that the only way to prove it is through using non-binary NAND-clauses.

Let us again consider the graph from Figure 2.24, shown again here for convenience:

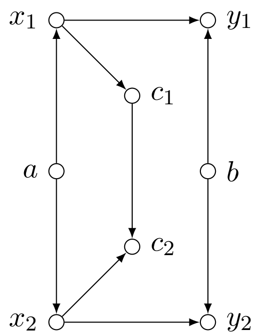

Figure 3.6

As before, the vertices y_1, y_2 and c_2 are initial vertices of disjoint rays, and not sinks.

The NAND-clause we show not to be binary-provable is $\overline{ab}$. Figure 2.25 proves $\overline{ab}$, but the proof contains two non-binary NAND-clauses. We will show that this is unavoidable. In order to do this, we utilize the table from Figure 2.27. We show it again here for convenience.

	a	b	x_1	x_2	y_1	y_2	c_1	c_2
a	α_1				α_1	α_1	α_1	α_2
b	–	α_4	α_4	α_3			α_3	α_4
x_1	–	–	α_4			α_6		α_6
x_2	–	–	–	α_5	α_5		α_5	
y_1	–	–	–	–	α_1	α_1	α_1	α_2
y_2	–	–	–	–	–	α_1	α_1	α_2
c_1	–	–	–	–	–	–	α_1	
c_2	–	–	–	–	–	–	–	α_4

Figure 3.7

Suppose there is a rule application with all binary NAND-clauses in the premise and with $\overline{ab}$ in the conclusion. Based on the (Rneg)-rule, we know that the premise must contain at least one binary NAND-clause containing a and at least one containing b. The above table tells us that the only provable binary NAND-clauses that contain a or b are the ones in the axiom set: $\overline{ax_1}$, $\overline{ax_2}$, $\overline{by_1}$ and $\overline{by_2}$. Since we do not want any x_i or y_i in the conclusion, these variables have to be removed by the OR-clause. The OR-clauses $x_1y_1c_1$ and $x_2y_2c_2$ are the only ones that contain both an x and a y, making them the only OR-clauses that can potentially conclude with $\overline{ab}$.

This gives us the following information: since both the possible OR-clauses are of length 3, the rule application concluding with $\overline{ab}$ has 3 NAND-clauses in its premise; one containing an x-vertex (either x_1 or x_2), one containing a y-vertex and one containing a c-vertex. Looking at our table again, we see that the potentially provable NAND-clauses containing a c-vertex are again the axioms only. Since there are no provable NAND-clauses on the form ac_i or bc_i, we get that the conclusion of our rule cannot possibly be of length 2, contradicting our assumption. We can therefore conclude that $\overline{ab}$ is *not* binary-derivable, thus disproving Conjecture 21.

3.3 Unary NAND-clauses in graph theories

The fact that some binary NAND-clauses are not binary-derivable will now be used to show that some *unary* NAND-clauses are not binary-derivable.

We use our graph from Figure 3.6 to form the following, bigger graph:

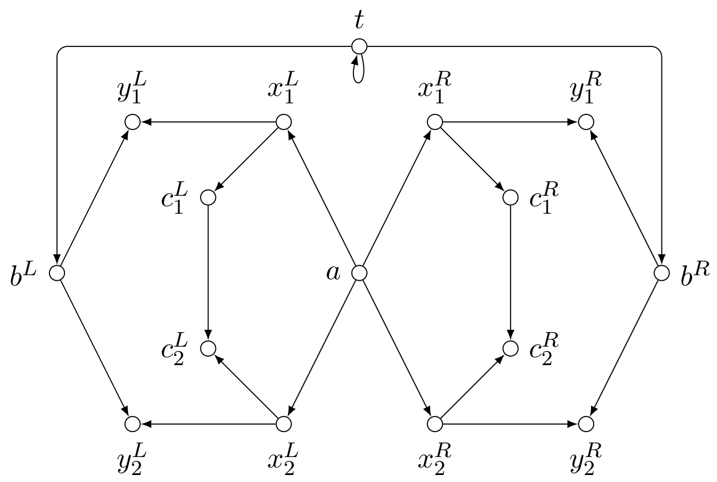

Figure 3.8

The above graph contains two copies of the graph from Figure 3.6, only connected by their shared vertex a and the vertex t that has both b^L and b^R in its neighborhood. We will refer to the two copies as the left component and the right component, with a being in both and t being in none.

The rest of this section will show that the unary NAND-clause $\overline{a}$ is provable, but not binary derivable.

Both $\overline{ab^L}$ and $\overline{ab^R}$ can be proven in the same manner as $\overline{ab}$ was proven earlier in Figure 2.25. This makes the proof of $\overline{a}$ trivial, as shown in Figure 3.9:

$$\dfrac{\bar{t} \qquad \dfrac{\cdots}{\overline{ab^L}} \qquad \dfrac{\cdots}{\overline{ab^R}}}{\bar{a}}\ tb^Lb^R$$

Figure 3.9

To show that $\bar{a}$ is not binary-derivable, we will first prove the following lemma:

Lemma 22. *Based on the graph in Figure 3.8, the only binary-derivable, non-axiomatic binary NAND-clause containing the atom a is $\overline{at}$.*

Proof. First, the following Neg-proof shows that $\overline{at}$ is indeed binary-derivable.

$$\dfrac{\overline{ax_1^L} \qquad x_2^Ly_2^Lc_2^L\ \dfrac{b^Ly_1^Ly_2^L\ \dfrac{tb^Rb^L\ \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^Ly_1^L}}{\overline{ty_1^L}} \quad \overline{tb^L} \quad \overline{x_2^Ly_2^L}}{\overline{tx_2^L}} \quad \overline{c_1^Lc_2^L} \quad \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^Ly_2^L}}{\overline{ty_2^L}}\ tb^Rb^L}{\overline{tc_1^L}} \qquad \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^Ly_1^L}}{\overline{ty_1^L}}\ tb^Rb^L}{\overline{at}}\ x_1^Ly_1^Lc_1^L$$

Figure 3.10

We now show that $\overline{at}$ is the only one. This is done by structural induction on the complexity of the proof tree for some binary NAND-clause $\overline{a\gamma}$, showing that γ always has to equal either some x-vertex or t.

In the base case, the proof tree is just an axiom. The only axiomatic $\overline{a\gamma}$ are the ones where γ is an x-vertex, so the lemma holds.

For the inductive step, suppose we have a proof that concludes with some binary NAND-clause $\overline{a\gamma}$; we have that the premise of the last proof step must contain at least one NAND-clause containing a and at least one containing γ.

We introduce a table similar to the one in Figure 2.27, showing what pairs of vertices from the graph in Figure 3.8 can be 1 under the same solution and thus correspond to binary NAND-clauses unprovable in Neg. Clauses in cells with an X are unprovable. Since the property is symmetric, the lower half of the table is left out.

	a	b^L	x_1^L	x_2^L	y_1^L	y_2^L	c_1^L	c_2^L	b^R	x_1^R	x_2^R	y_1^R	y_2^R	c_1^R	c_2^R	t
a																
b^L	–	X	X	X			X	X	X	X	X	X	X	X	X	
x_1^L	–	–	X			X		X	X	X	X	X	X	X	X	
x_2^L	–	–	–	X	X		X		X	X	X	X	X	X	X	
y_1^L	–	–	–	–	X	X	X	X	X	X	X			X	X	
y_2^L	–	–	–	–	–	X	X	X	X	X	X			X	X	
c_1^L	–	–	–	–	–	–	X		X	X	X	X	X	X	X	
c_2^L	–	–	–	–	–	–	–	X	X	X	X	X	X	X	X	
b^R	–	–	–	–	–	–	–	–	X	X	X			X	X	
x_1^R	–	–	–	–	–	–	–	–	–	X			X		X	
x_2^R	–	–	–	–	–	–	–	–	–	–	X	X		X		
y_1^R	–	–	–	–	–	–	–	–	–	–	–	X	X	X	X	
y_2^R	–	–	–	–	–	–	–	–	–	–	–	–	X	X	X	
c_1^R	–	–	–	–	–	–	–	–	–	–	–	–	–	X		
c_2^R	–	–	–	–	–	–	–	–	–	–	–	–	–	–	X	
t	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	

Figure 3.11

Our induction hypothesis gives us that if the NAND-clause containing a in the premise is non-axiomatic, then it must be $\overline{at}$. All NAND-clauses containing a in the premise must therefore be either $\overline{at}$ or one of the four axioms on the form $\overline{ax_i^K}$ (K being either L or R and i either 1 or 2).

Let us first consider the case where some $\overline{ax_i^K}$ is in the premise. Figure 3.12 illustrates the situation.

$$\cfrac{\overline{ax_i^K} \qquad \cfrac{\cdots}{\overline{\gamma y_i^K}} \qquad \cfrac{\cdots}{\overline{\gamma c_i^K}}}{\overline{a\gamma}} \; x_i^K y_i^K c_i^K$$

Figure 3.12

The only applicable OR-clause is the corresponding $x_i^K y_i^K c_i^K$, so the premise must contain two additional NAND-clauses, one containing y_i^K and one containing c_i^K. Because of our induction hypothesis, those two cannot contain a, so either both contain γ or one is unary while the other contain γ. Table 3.11 gives us that neither $\overline{y_i^K}$ nor $\overline{c_i^K}$ is provable, so they both have to contain γ. Therefore, the two additional NAND-clauses must be on the form $\overline{\gamma y_i^K}, \overline{\gamma c_i^K}$. Except for the case where $\gamma = x_i^K$, the only γ-substitution that makes both of these NAND-clauses provable is $\gamma = t$, giving us $\overline{at}$ in the conclusion. We again see this from Table 3.11.

The other case is where $\overline{at}$ appears in the premise. Figure 3.13 illustrates this situation.

$$\dfrac{\dfrac{\cdots}{\overline{at}} \qquad \dfrac{\cdots}{\overline{\gamma b^L}} \qquad \dfrac{\cdots}{\overline{\gamma b^R}}}{\overline{a\gamma}}\ tb^L b^R$$

Figure 3.13

In order to "strip off" the t, the only applicable OR-clause is $tb^L b^R$, again giving us three NAND-clauses in the premise. The two additional clauses must contain b^L and b^R, respectively. From Table 3.11, we get that none of them are provably false, and none of them can contain a because of the induction hypothesis, so they both must contain some γ. The only γ-substitution applicable such that both $\overline{\gamma b^L}$ and $\overline{\gamma b^R}$ is provable is, again, $\gamma = t$, giving us $\overline{at}$ in the conclusion.

The only binary-provable, non-axiomatic, binary NAND-clause containing a is therefore $\overline{at}$. $\square$

We can now easily prove the following lemma:

Lemma 23. *Given the graph from Figure 3.8, $\overline{a}$ is not binary-derivable.*

Proof. Suppose we have a binary proof concluding with $\overline{a}$. There must exists a NAND-clause in the immediate premise of this conclusion containing either $\overline{at}$ or some $\overline{ax_i^K}$. Because of Lemma 22, we know that no other binary NAND-clauses containing a are binary-derivable.

In the case with $\overline{at}$, the OR-clause $tb^L b^R$ must be used. Now, in order to prove $\overline{a}$, the premise must contain either $\overline{ab^L}$ or $\overline{b^L}$, but $\overline{ab^L}$ is not binary-derivable and $\overline{b^L}$ is not even provable.

In the case with $\overline{ax_i^K}$ in the premise, we get the same problem; the only applicable OR-clause is $x_i^K y_i^K c_i^K$, but neither $\overline{ay_i^K}$ nor $\overline{y_i^K}$ is binary-derivable.

We therefore get that no binary proof exists with $\overline{a}$ as its conclusion, making $\overline{a}$ provable, but not binary-derivable. $\square$

We will use this result in the following section to show that there are inconsistent graphs such that the inconsistency is not binary-derivable.

3.4 Inconsistencies in graph theories

In this section we consider the graph in Figure 3.14. Any mention of vertices, edges and components in this section will refer to this graph.

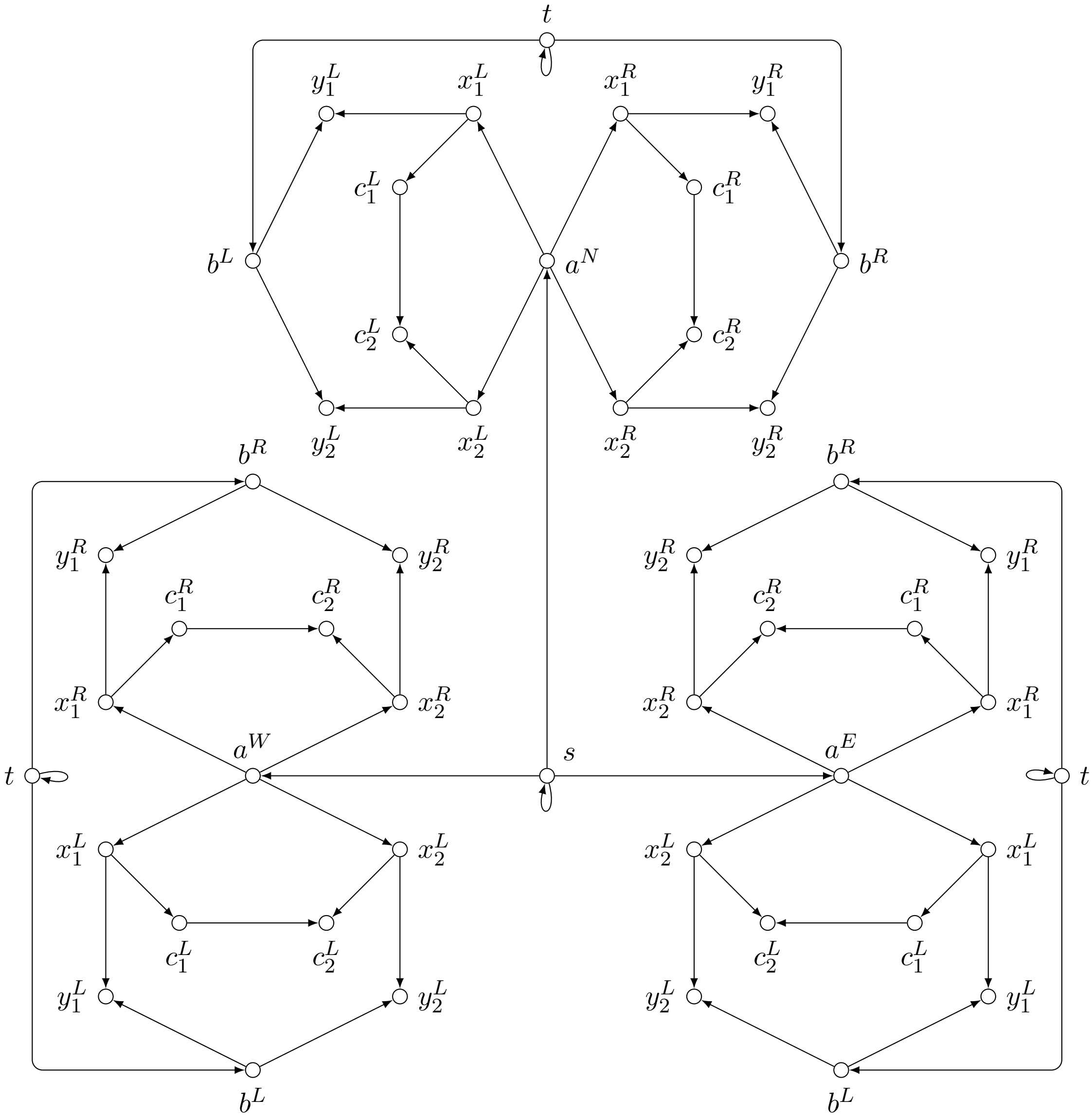

Figure 3.14

The graph contains 3 copies of the graph from Figure 3.8; one northern (N), one western (W) and one eastern (E). We call these *components*. Every vertex in the graph is contained within exactly one component, except the vertex s which is not inside any component.

The first thing to notice is that the graph is inconsistent; the proof of $\overline{a}$ from Figure 3.9 can be applied to each component in the above graph, giving us the provability of $\overline{a^N}$, $\overline{a^W}$ and $\overline{a^E}$. From here, the inconsistency proof is trivial; shown in Figure 3.15.

$$\dfrac{\overline{s} \qquad \dfrac{\cdots}{\overline{a^N}} \qquad \dfrac{\cdots}{\overline{a^W}} \qquad \dfrac{\cdots}{\overline{a^E}}}{\varnothing}\; sa^Na^Wa^E$$

Figure 3.15

We will ultimately show that the inconsistency is not binary-derivable, but first we prove some lemmata.

Lemma 24. *Let the vertices u and v be from different components in the graph. If $\overline{uv}$ is binary-derivable, then its proof contains either $\overline{a^N}$, $\overline{a^W}$ or $\overline{a^E}$.*

Proof. We prove the lemma by structural induction over the complexity of the proof tree.

Base Case: No axiom $\overline{uv}$ exists such that u and v are vertices from different components, so our claim vacuously holds.

Induction Step: Suppose we have a binary proof of $\overline{uv}$ where u and v are from different components. The premise of the last proof step must contain at least one binary NAND-clause containing u and one containing v.

$$\dfrac{\overline{u\ldots} \qquad \overline{v\ldots} \qquad \ldots}{\overline{uv}}\; Y$$

Figure 3.16

If either u or v are in a clause together with a vertex from a component different from their own, we get from the induction hypothesis that their proof must contain either $\overline{a^N}$, $\overline{a^W}$ or $\overline{a^E}$, so we are done.

Otherwise, u and v are each either in a clause together with the vertex s or a vertex from their own component. Each of these cases implies $Y = sa^Na^Wa^E$ in the last proof step, since it is the only OR-clause containing s and the only OR-clause containing vertices from different components. The premise therefore contains two additional NAND-clauses; one of them containing the a-vertex of the component not containing u or v – let us call this vertex a^t.

Figure 3.17 illustrates the two possible cases.

If the clause containing a^t is unary, we are done. If it is binary, it must either contain u or v in order to give $\overline{uv}$ in the conclusion, and since a^t is in a component different from both u and v, the induction hypothesis gives us the claim.

$$\frac{\overline{s\ldots} \quad \overline{ua^N} \quad \overline{va^W} \quad \overline{a^E\ldots}}{\overline{uv}}\ sa^Na^Wa^E \qquad \frac{\overline{us} \quad \overline{va^N} \quad \overline{a^W\ldots} \quad \overline{a^E\ldots}}{\overline{uv}}\ sa^Na^Wa^E$$

Figure 3.17

The proof must therefore contain either $\overline{a^N}$, $\overline{a^W}$ or $\overline{a^E}$. □

Lemma 25. *$\overline{a^N}$, $\overline{a^W}$ and $\overline{a^E}$ are not binary-derivable.*

Proof. We prove it by structural induction over the complexity of the proof tree.

Base Case: Neither $\overline{a^N}$, $\overline{a^W}$ nor $\overline{a^E}$ are axioms, so the claim vacuously holds.

Induction step: Suppose we have a binary proof of $\overline{a}$, where a is either a^N, a^W or a^E. The vertices within the same component as the one containing a will be referred to as *internal* vertices, while the vertices within the two other components will be referred to as *external* vertices. Note that since s is not inside any component, it is neither internal nor external.

First, since the proof is binary, we get from our induction hypothesis that neither $\overline{a^N}$, $\overline{a^W}$ nor $\overline{a^E}$ appears in it.

We get from Lemma 23 that $\overline{a}$ is not binary-derivable if one is using internal vertices only, so the binary proof must use vertices outside the component containing a; either the s-vertex or external vertices.

If it uses s, consider the last proof step with s in the premise. The OR-clause used in this proof step must be $sa^Na^Wa^E$, being the only OR-clause containing s and thus the only clause able to remove it. This OR-clause contains 4 vertices, so the premise must contain 3 additional NAND-clauses; one containing a^N, one containing a^W and one containing a^E.

We get the following restrictions on these 3 NAND-clauses.

- None of them can be unary, from the induction hypothesis.
- None of them can be binary and contain s, since that would give an s in the conclusion, contradicting our assumption of this being the last proof step with s in the premise.
- None of them can be binary and contain vertices from two different components, from Lemma 24 the induction hypothesis.

The 3 additional NAND-clauses must therefore all contain a second vertex p from their own component.

Figure 3.18 illustrates the situation.

$$\frac{\overline{s\ldots} \quad \overline{a^N p^N} \quad \overline{a^W p^W} \quad \overline{a^E p^E}}{\overline{p^N p^W p^E}} \; sa^N a^W a^E$$

Figure 3.18

Since the three components are disjoint, these three p-vertices are different, making the conclusion of the proof step non-binary, contradicting our original assumption of the proof being binary.

If the proof does *not* contain s, then it uses external vertices. Consider the last proof step containing external vertices in the premise. This proof step removes these vertices and conclude with some clause only internal vertices. The premise must contain, in addition to all the clauses with external vertices, at least one binary NAND-clause with an internal vertex, in order to make the conclusion non-empty. This clause cannot contain any external vertices, from Lemma 24 and the induction hypothesis, so it must be a clause with two internal vertices. The OR-clause used in the proof step must therefore contain both external vertices and at least one internal vertex. Since $sa^N a^W a^E$ is the only OR-clause containing vertices from different components, it is the only option, but we just assumed that s is not in this proof, so we reach a contradiction.

$\overline{a}$ is thus not binary-derivable. □

Corollary 26. *If the vertices u and v are from different components in the graph, $\overline{uv}$ is not binary-derivable.*

Proof. We get this directly from Lemma 24 and Lemma 25. □

Lemma 27. *Any proof of inconsistency must contain the NAND-clause $\overline{s}$.*

Proof. Consider the graph G. If one removes the loop on the vertex s, the graph is no longer inconsistent; each a-vertex can be assigned 0 and s can be assigned 1. Formally, we have that $G' = G \setminus E(s,s) \nvdash \varnothing$. Because of soundness of Neg, this gives us that $\mathcal{T}(G') = \mathcal{T}(G) \setminus \{\overline{s}\} \nvdash \varnothing$ (recall the $\mathcal{T}$-notation from Section 1.4).

Suppose there is a proof of $\mathcal{T}(G) \vdash \varnothing$ that does not use $\overline{s}$; this proof will also be a proof of $\mathcal{T}(G') \vdash \varnothing$ violating the fact that Neg is sound. □

Theorem 28. *$\varnothing$ is not binary-derivable.*

Proof. Assume $\varnothing$ is binary-derivable. We have from Lemma 27 that the proof must contain $\overline{s}$. Consider the last proof step with s in the premise. The OR-clause $sa^N a^W a^E$ is again the only clause that can do this, being the only OR-clause containing s. The premise thus contains 3 additional NAND-clauses; one containing a^N, one containing a^W and one containing a^E. We get the following restrictions on these three clauses:

- None of them can be unary, from Lemma 25.

- None of them can be binary and contain s, since that would give an s in the conclusion, contradicting our assumption of this being the last proof step with s in the premise.
- None of them can be binary and contain vertices from two different components, from Corollary 26.

The 3 additional NAND-clauses must therefore all contain a second vertex from their own component, making the conclusion non-binary, contradicting our original assumption of the proof being binary.

$\varnothing$ is therefore not binary-derivable. $\square$

3.5 Additional findings

While developing the proof of the refutational incompleteness of BNeg, some additional observations where made along the way. Even though they did not ultimately turn out useful for the main proof, they do bear some general significance. These observations are presented and proved in this section.

3.5.1 Isolated components

Given a graph $\mathbf{G} = \langle G, N \rangle$ and a set of vertices $A \subseteq G$, if $N(A) \subseteq A$ and $N^{-}(A) \subseteq A$, we call it an *isolated component* of the graph $\mathbf{G}$.

Since there are no edges between any two isolated components, we get that no axiomatic NAND- and OR-clauses can contain vertices from two different isolated components.

We now prove the following lemma:

Lemma 29. *Given a graph $\mathbf{G} = \langle G, N \rangle$ and a set of vertices $B \subseteq G$; if $\overline{B}$ is provable in Neg, then all the vertices in B are from the same isolated component.*

An alternative formulation of Lemma 29 is that the vertices in B must be connected in the underlying undirected graph.

Proof. We prove it by structural induction over the complexity of the proof tree of some NAND-clause $\overline{B}$.

Since no axioms contain vertices from different isolated components, the claim holds for the base case.

For the inductive step, suppose we have a proof of the NAND-clause $\overline{B}$. The induction hypothesis lets us assume that each of the NAND-clauses in the premise contain vertices from the same component. Since no OR-clause contain vertices from different components, we get that *all* the NAND-clauses in the premise are from the same component, so the concluding $\overline{B}$ must also contain vertices from the same component.

Any provable $\overline{B}$ must therefore contain vertices from the same component. $\square$

3.5.2 Components connected by a common source

Let $\mathbf{G} = \langle G, N \rangle$ be a digraph with a source vertex x. Let $\{A_i\}$ be a *partitioning* of $G \setminus \{x\}$ (a set of pairwise disjoint, nonempty subsets of G such that $\bigcup\{A_i\} = G \setminus \{x\}$) such that for each A_i we have that $N(A_i) \subseteq A_i$ and $N^{-}(A_i) \setminus A_i = \{x\}$. In words, nothing points out of A_i and only x points in.

The induced subgraphs we get from each A_i will be referred to as *components*.

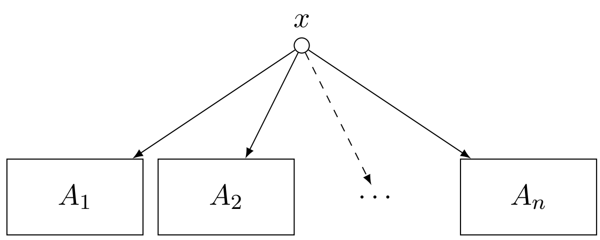

Figure 3.19: Graph G

A couple of observations can be made based on the above graph-construction G:

- The vertex x only appears in 1 OR-clause, since it is a source vertex. This is also the only OR-clause containing vertices from different components. We will call this OR-clause X and formally define it as $N(x) \cup \{x\}$.
- For any vertex p except x, we have that p only appears in axiomatic NAND-clauses together with either x or other vertices in the same component.

Based on graph G from Figure 3.19 we show the following lemma:

Lemma 30. *Given a set of vertices $B \subseteq \bigcup\{A_i\}$ such that $\forall A \in \{A_i\} : B \not\subseteq A$; if $G \vdash_{Neg} \overline{B}$ then the proof must contain the OR-clause X.*

Proof. The lemma will be proven by structural induction on the complexity of the proof tree for $\overline{B}$.

In the base case, the proof tree is just an axiom. There exists no axiom in $\bigcup A_i$ which is not a subset of some component A_k, so our claim vacuously holds.

For the inductive step, suppose we have a proof of $\overline{B}$ and consider the premise of the last rule application. The following figure illustrates the general situation, with $B = B_1 \cup B_2 \cup \cdots \cup B_n$:

$$\dfrac{\dfrac{\cdots}{\overline{B_1c_1}} \qquad \dfrac{\cdots}{\overline{B_2c_2}} \qquad \cdots \qquad \dfrac{\cdots}{\overline{B_nc_n}}}{\overline{B}} \; c_1c_2\ldots c_n$$

Figure 3.20

Since $\overline{B}$ does not contain x, neither does any of the B_i-clauses. If some $c_i = x$, then the OR-clause $c_1c_2\ldots c_n = X$, being the only OR-clause containing x.

Otherwise, if some NAND-clause $\overline{B_ic_i}$ contains vertices from two different components, the induction hypothesis tells us that its proof must contain X, so we are done.

The last case is where each NAND-clause $\overline{B_ic_i}$ in the premise contains vertices from one component only. Since $\overline{B}$ contains vertices from at least two different components, it cannot be the case that all the NAND-clauses in the premise take their vertices from the same component. At least two of the c_i-vertices therefore come from different components, so the OR-clause used must be X, being the only OR-clause containing vertices from different components.

The proof of $\overline{B}$ must therefore contain the OR-clause X. $\square$

Chapter 4

Conclusion and future work

This thesis has found various graph structures implying provability of certain NAND-clauses in Neg. The process of continuously generalizing these structures has given several examples of unconventional graphs still providing certain provable clauses. One of these exemplified that some provable binary NAND-clauses are not binary-derivable. This example was further extended to show that Neg is *not* refutationally complete when restricted to using binary NAND-clauses only.

In our context, this is primarily a negative result. If BNeg *was* refutationally complete, our search for a graph-structural equivalent of provable clauses would be easier, allowing us to assume that any clause is provable using binary NAND-clauses only.

Knowing this, one can move forward by trying to find other features of Neg. One example is its non-explosiveness, mentioned in [9]. While classical proof systems are explosive and thus able to prove anything from inconsistent theories, Neg can only prove certain clauses. It would be interesting to take a closer look at these clauses that are provable by the virtue of the graph being inconsistent, and potentially develop some definition of the "explosive range" of a graph. Coming back to our original interest in paradoxes, this property becomes useful in attempts to identify and isolate the part of a theory that makes it paradoxical. The concept of *local kernels*, as defined and studied in [2], might contribute to this definition of an explosive range.

The non-explosiveness of Neg comes from the more general fact that Neg does not have *weakening*, i.e. while rules like $\Gamma \vdash x \;\; \Rightarrow \;\; \Gamma \vdash x \vee y$ are admissible in classical proof systems, this is not the case for Neg.

Since Neg is not complete, it does not hold in general that $\Gamma \vDash \overline{A} \Rightarrow \Gamma \vdash \overline{A}$. An interesting question might therefore be "*when* does it hold?". Corollary 5.1 from [9] tells us that $\Gamma \vDash \overline{A} \Leftrightarrow \exists B \subseteq A : \Gamma \vdash \overline{B}$. A direct consequence of this is that $\Gamma \vDash \overline{A} \Rightarrow \Gamma \vdash \overline{A}$ holds when $\forall B \subset A : \Gamma \nvdash B$ and by soundness also when $\forall B \subset A : \Gamma \nvDash B$. We can formalize this into a corollary on its own:

Corollary 31. *Given a graph* $\mathbf{G} = \langle G, N \rangle$ *and a set of vertices* $A \subseteq G$*, let* $\Gamma = \mathcal{T}(\mathbf{G})$*:*

$$\Gamma \vDash \overline{A} \wedge \forall B \subset A : \Gamma \nvDash \overline{B} \;\Rightarrow\; \Gamma \vdash \overline{A} \tag{4.1}$$

In words, if no kernel in G contains A, but for each subset $B \subset A$, there is a kernel that contains B, then $\overline{A}$ is provable in Neg.

Based on this observation, one might be able to find some interesting relations between the kernels in the graph and the set A.

Also, more work could be put into exploring graph structural relations V such that for a graph and a subset A of its vertices, we have $\Gamma \vdash \overline{A} \Rightarrow V(A)$. The relation "being connected in the underlying undirected graph" is certainly such a relation, as shown in Section 3.5.1. If one is able to strengthen that relation, it will probably make a bigger contribution to a potential proof of Conjecture 9 (from Section 1.6) than relations satisfying the inverse implication.

Appendix A

Proofs

A.1 Translating CNF to GNF

Since any PL theory can be expressed in CNF, showing that any theory P in CNF can be translated to a theory R in GNF such that P and R are equisatisfiable gives us that any PL theory has an equisatisfiable GNF.

Given any CNF theory P, for each formula in it, follow the steps below to acquire its corresponding GNF theory.

Step 1: For each atom x_i in the formula, introduce a fresh variable x_i' and add the following two GNF formulae to P: $x_i' \leftrightarrow \neg x_i, x_i \leftrightarrow \neg x_i'$, (unless this has already been done while translating an earlier formula in the theory).

Step 2: In each clause of the formula, replace every negative literal $\neg x_i$ with its corresponding x_i' from step 1. Every clause in the formula does now contain positive literals only.

Step 3: For every clause $(x_1 \vee x_2 \vee \cdots \vee x_n)$, replace it with the following formula, where y is fresh:

$$y \leftrightarrow (\neg x_1 \wedge \neg x_2 \wedge \cdots \wedge \neg x_n \wedge \neg y)$$

This formula is equisatisfiable with our original clause.

The combined set of all these acquired formulae will make up our GNF-theory. We have only added proper GNF formulae and all variables appear to the left in exactly one clause, so R will indeed be a GNF theory.

Step 1 is adding a bi-implication formula that is always satisfiable, since one of two variables in it is fresh. One can think of it as a labelling of an already existing variable. Thus, adding these formulae to a theory does not change its satisfiability/non-satisfiability.

Step 2 is replacing each negative literal with its label, also not changing the satisfiability.

Step 3 is replacing one formula with an equisatisfiable formula, thus not changing the overall satisfiability.

Since none of the steps above change the satisfiability of the theory, we can conclude that our acquired theory is equisatisfiable with to original CNF theory.

Below are two examples of CNF-theories with their corresponding GNF-theories.

Example 32.

$$T1_{\text{CNF}}: \quad (a \vee b) \tag{A.1}$$

$$T1_{\text{GNF}}: \quad (a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (b \leftrightarrow \neg b'), (b' \leftrightarrow \neg b), (y_1 \leftrightarrow (\neg a \wedge \neg b \wedge \neg y_1)) \tag{A.2}$$

$$T2_{\text{CNF}}: \quad (\neg a) \tag{A.3}$$

$$T2_{\text{GNF}}: \quad (a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (y_1 \leftrightarrow (\neg a' \wedge \neg y_1)) \tag{A.4}$$

A.2 Inconsistency of Stretched Yablo

One variation of the Yablo-graph is the Stretched Yablo-graph, presented by Walicki in [9]. While the Yablo-graph is a ray where each vertex on it has direct edges to all the vertices after it, the Stretched Yablo-graph is a ray where each vertex has *disjoint paths* (with certain properties depending on the variation of Stretched Yablo) to all vertices in after it.

The variation on which we will be proving an inconsistency is one with a set of core vertices $\mathbb{N}$ where each vertex x has a disjoint path to every vertex y after it, such that the length of the path is $(2 \times (y - x) - 1)$.

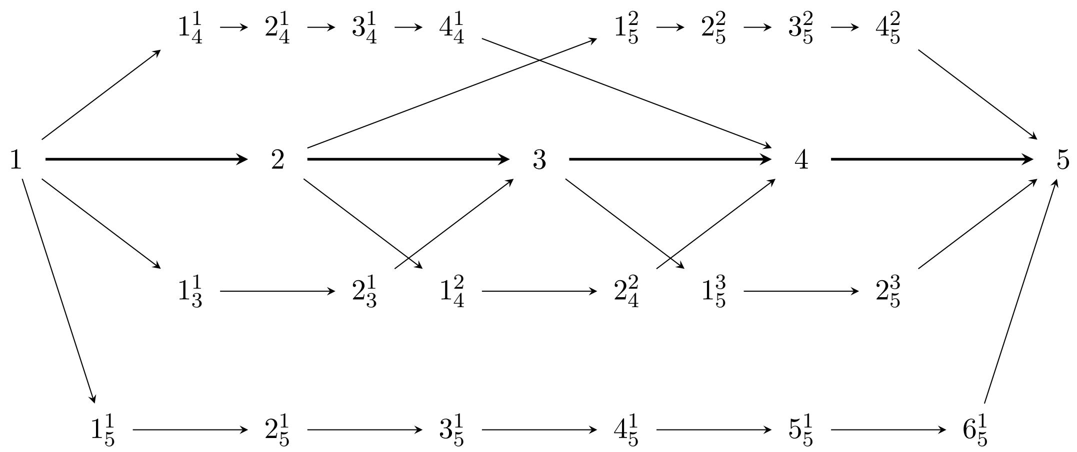

Figure A.1: Stretched Yablo

Shown above are parts of the Stretched Yablo-graph described. We denote the core vertices by natural numbers. The peripheral vertices contained in each path is denoted by n^x_y where x and y is the source and target, respectively, of the path in which the vertex is contained. n denoted the relative position on the path.

Note that since two core vertices x and y have a peripheral path between them of length $L^x_y = 2(y-x)-1$ (from the definition above), each path consists of $L^x_y - 1 = 2(y-x)-2$ peripheral vertices. The vertex pointing back onto the core ray will thus alway be the vertex $(2(y-x)-2)^x_y$.

Based on the given definition of Stretched Yablo and the notation from above, we get the following sets of axioms:

$$N1a = \{\ \overline{x(x+1)} \mid x \in \mathbb{N}\ \} \tag{A.5}$$
$$N1b = \{\ \overline{x1^x_y} \mid x, y \in \mathbb{N}, x+1 < y\ \} \tag{A.6}$$
$$N2 = \{\ \overline{n^x_y(n+1)^x_y} \mid n, x, y \in \mathbb{N}, x+1 < y, n < 2(y-x)-2\ \} \tag{A.7}$$
$$N3 = \{\ \overline{n^x_y y} \mid n, x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2\ \} \tag{A.8}$$
$$O1 = \{\ x(x+1)1^x_{x+2}1^x_{x+3}\dots \mid x \in \mathbb{N}\ \} \tag{A.9}$$
$$O2 = \{\ n^x_y(n+1)^x_y \mid n, x, y \in \mathbb{N}, x+1 < y, n < 2(y-x)-2\ \} \tag{A.10}$$
$$O3 = \{\ (n^x_y y \mid n, x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2\ \} \tag{A.11}$$

$$\text{NAND} = N1a \cup N1b \cup N2 \cup N3 \qquad \text{OR} = O1 \cup O2 \cup O3 \tag{A.12}$$

One can identify the different axioms by looking at the figure above[1]: $N1a$ are the edges making up the core ray of the graph, $N1b$ are the edges going from a core vertex onto a peripheral vertex, $N2$ are the edges between two peripheral vertices and $N3$ are the edges going from a peripheral vertex back onto a core vertex. $O1$ are the vertices on the core ray, $O2$ and $O3$ are the peripheral vertices, $O3$ being the ones that points back to a core vertex.

In order to prove $\varnothing$ from these axiom, we first prove three sets of intermediate clauses:

$$D1 = \{\ \overline{xy} \mid x, y \in \mathbb{N}, x \neq y\ \} \tag{A.13}$$
$$D1b = \{\ \overline{1^x_y n^x_y} \mid x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2\ \} \tag{A.14}$$
$$D2 = \{\ \overline{x1^y_z} \mid x, y, z \in \mathbb{N}, x \neq z, y+1 < z\ \} \tag{A.15}$$
$$D3 = \{\ \overline{1^1_x 1^y_z} \mid x, y, z \in \mathbb{N}, x \neq z, y+1 < z\ \} \tag{A.16}$$

We already have $\overline{xy}$ from our axioms whenever $y = x \pm 1$, so let us assume that either $y < x-1$ or $x+1 < y$. In both these cases, we get that there exists a peripheral path between x and y. Let us – without loss in generality – assume that $x+1 < y$. We thus have a path from x to y containing $n = 2 \times (y-x) - 2$ peripheral vertices. Consider now the following proof:

[1] recall how NAND-clauses corresponds to edges, while OR-clauses corresponds to vertices

$$
\text{(O2) } 1^x_y2^x_y \; \dfrac{\overset{(N1b)}{\overline{x1^x_y}} \qquad \overset{(N2)}{\overline{2^x_y3^x_y}}}{\overline{x3^x_y}}
$$

$$
\text{(O2) } 3^x_y4^x_y \; \dfrac{\overline{x3^x_y} \qquad \overset{(N2)}{\overline{4^x_y5^x_y}}}{\overline{x5^x_y}}
$$

$$
\text{(O2) } 5^x_y6^x_y \; \dfrac{\overline{x5^x_y} \qquad \overset{(N2)}{\overline{6^x_y7^x_y}}}{\overline{x7^x_y}}
$$

$$
\vdots
$$

$$
\text{(O2) } (n-1)^x_yn^x_y \; \dfrac{\overline{x(n-1)^x_y} \qquad \overset{(N3)}{\overline{n^x_yy}}}{\overline{xy}}
$$

Figure A.2: Proof, D1

Intuitively, this proof follows along the peripheral path between the two core vertices. It is important to notice that since all our peripheral paths are odd in length, $(n-1)$ is always odd. This guarantees that we are able to prove $\overline{x(n-1)^x_y}$ in $n/2$ steps using the strategy above.

Using x, y, n from above, the proof of D1b follows a similar pattern as D1:

$$
\text{(O2) } 2^x_y3^x_y \; \dfrac{\overset{(N2)}{\overline{1^x_y2^x_y}} \qquad \overset{(N2)}{\overline{3^x_y4^x_y}}}{\overline{1^x_y4^x_y}}
$$

$$
\text{(O2) } 4^x_y5^x_y \; \dfrac{\overline{1^x_y4^x_y} \qquad \overset{(N2)}{\overline{5^x_y6^x_y}}}{\overline{1^x_y6^x_y}}
$$

$$
\text{(O2) } 6^x_y7^x_y \; \dfrac{\overline{1^x_y6^x_y} \qquad \overset{(N2)}{\overline{7^x_y8^x_y}}}{\overline{1^x_y8^x_y}}
$$

$$
\vdots
$$

$$
\text{(O2) } (n-2)^x_y(n-1)^x_y \; \dfrac{\overline{1^x_y(n-2)^x_y} \qquad \overset{(N2)}{\overline{(n-1)^x_yn^x_y}}}{\overline{1^x_yn^x_y}}
$$

Figure A.3: Proof D1b

Just like with D1, it is crucial that the path is odd in order for this proof strategy to work.

As for the proof of D2, we again have a trivial case where $x = y$ making the clause simply an instance of $N1b$. We therefore continue under the assumption that $x \neq y$, together with the other restrictions from the definition of $D2$. Using the same notation as above, where n is the number of vertices in the peripheral path between y and z, we now get the following short proof:

$$(\mathrm{O3})\ n^y_z z \quad \frac{\overset{(D1)}{\overline{xz}} \qquad\qquad \overset{(D1b)}{\overline{1^y_z n^y_z}}}{\overline{x1^y_z}}$$

Figure A.4: Proof, D2

Using $D1$ and $D1b$ in the above proof restricts us to cases where $x \neq z$ and where $y + 1 < z$, but these are restrictions we have already assumed.

Lastly, we prove D3. n now denotes the number of vertices in the peripheral path between y and z.

$$(\mathrm{O3})\ n^y_z z \quad \frac{\overset{(D2)}{\overline{1^1_x z}} \qquad\qquad \overset{(D1b)}{\overline{1^y_z n^y_z}}}{\overline{1^1_x 1^y_z}}$$

Figure A.5: Proof, D3

We are again restricting our cases by using $D3$ and $D1b$, but these restrictions are already assumed.

We now have what we need to prove the inconsistency of Stretched Yablo. First, we prove the following NAND-clauses: $\overline{1}, \overline{2}, \overline{1^1_3}, \overline{1^1_4 1^1_5} \ldots$.

Proving $\overline{1}$:

$$(\mathrm{O1})\ 231^2_4 1^2_5 1^2_6 \ldots \quad \frac{\overset{(N1)}{\overline{12}} \qquad \overset{(D1)}{\overline{13}} \qquad \overset{(D2)}{\overline{11^2_4}} \qquad \overset{(D2)}{\overline{11^2_5}} \qquad \overset{(D2)}{\overline{11^2_6}} \qquad \ldots}{\overline{1}}$$

Figure A.6

Proving $\overline{2}$:

$$(\mathrm{O1})\ 341^3_5 1^3_6 1^3_7 \ldots \quad \frac{\overset{(N1)}{\overline{23}} \qquad \overset{(D1)}{\overline{24}} \qquad \overset{(D2)}{\overline{11^3_5}} \qquad \overset{(D2)}{\overline{11^3_6}} \qquad \overset{(D2)}{\overline{11^3_7}} \qquad \ldots}{\overline{2}}$$

Figure A.7

Proving $\overline{1^1_3}$:

$$\text{(O1) } 451^4_6 1^4_7 1^4_8 \ldots \quad \frac{\overset{(D2)}{\overline{1^1_3 4}} \qquad \overset{(D2)}{\overline{1^1_3 5}} \qquad \overset{(D3)}{\overline{1^1_3 1^4_6}} \qquad \overset{(D3)}{\overline{1^1_3 1^4_7}} \qquad \overset{(D3)}{\overline{1^1_3 1^4_8}} \qquad \ldots}{\overline{1^1_3}}$$

Figure A.8

Proving $\overline{1^1_4}$:

$$\text{(O1) } 561^5_7 1^5_8 1^5_9 \ldots \quad \frac{\overset{(D2)}{\overline{1^1_4 5}} \qquad \overset{(D2)}{\overline{1^1_4 6}} \qquad \overset{(D3)}{\overline{1^1_3 1^5_7}} \qquad \overset{(D3)}{\overline{1^1_3 1^5_8}} \qquad \overset{(D3)}{\overline{1^1_3 1^6_9}} \qquad \ldots}{\overline{1^1_4}}$$

Figure A.9

The clauses $\overline{1^1_4}$, $\overline{1^1_5}$, $\overline{1^1_5}$, $\overline{1^1_7}$, ... can be proven in exactly the same manner as above. With the results from above, we can finally prove $\varnothing$:

$$\text{(O1) } 121^1_3 1^1_4 1^1_5 \ldots \quad \frac{\overline{1} \qquad \overline{2} \qquad \overline{1^1_3} \qquad \overline{1^1_4} \qquad \overline{1^1_5} \qquad \ldots}{\varnothing}$$

Figure A.10

Having proved $\varnothing$ in Neg, we get from soundness of Neg that our Stretched Yablo-graph is indeed inconsistent.

A.3 Provability of NAND-clauses from vels

This section will prove the following statement: Given a graph where two vertices a and b are connected by a vel, as defined in Definition 14, Section 2.4, the binary NAND-clause $\overline{ab}$ is provable in Neg.

Let $\mathbf{G} = (G, N)$ be a graph containing the vertices a, b such that they have a vel between them. By definition, this means that there exists a vertex c such that there is an oddly trimmed path from a to c and from b to c, one of odd length and one of even length.

Let P and Q be the two sets containing the vertices in the path from a to c and from b to c, respectively. We will denote each element of P as p_i where $i \in \mathbb{N}$ is the position of that vertex in the path, starting at 0. The elements of Q will be named q_i by the same rule. We immediately have that $a = p_0$ and $b = q_0$. As long as the trimming restrictions are met, P and Q might overlap, i.e. we might have cases where $p_j = q_k$ from some i and j.

We assume, without any loss of generality, that the path from a to c is of odd length, making the path from b to c. We denote the lengths of the two paths by the numbers n and m, giving us that $c = p_n = q_m$. The path from b to c might also be empty, making $b = p_0 = c$.

This general variant of a vel can be illustrated in the following way, where the possibly branching vertices are shown with dashed edges out of them.

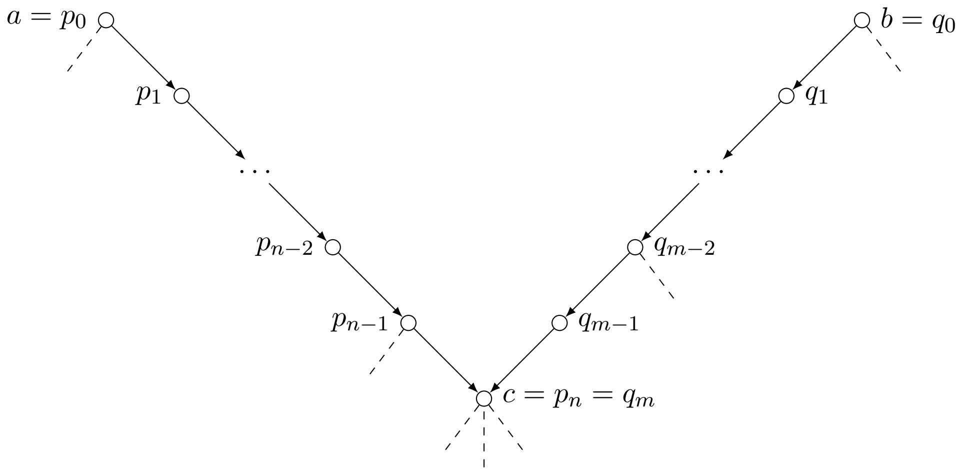

Figure A.11: A general vel between a and b

Let NAND_G and OR_G denote the axiomatic NAND- and OR-clauses we get from our graph. We can work out the following subset of NAND_G:

$$\text{NAND}_V = \{\ \overline{p_i p_{i+1}} \mid 0 \le i < n\ \} \cup \{\ \overline{q_j q_{j+1}} \mid 0 \le j < m\ \} \tag{A.17}$$

Since both paths are oddly trimmed, every vertex at an odd position in its path will only have one out-edge, resulting in a binary OR-clause. This makes us able to work out the following subset of OR_G:

$$\mathrm{OR}_V = \{\ p_ip_{i+1} \mid 0 \leq i < n, i \text{ is odd}\ \} \cup \{\ q_jq_{j+1} \mid 0 \leq j < m, j \text{ is odd}\ \} \tag{A.18}$$

The proof of $\overline{ab}$ can now be worked out, based only on the axioms $\Gamma_V = \mathrm{NAND}_V \cup \mathrm{OR}_V$:

$$\dfrac{q_{m-1}c \quad \dfrac{p_{n-2}p_{n-1} \quad \dfrac{\dfrac{p_3p_4 \quad \dfrac{p_1p_2 \quad \overline{ap_1} \quad \overline{p_2p_3}}{\overline{ap_3}} \quad \overline{p_4p_5}}{\overline{ap_5}}}{\dfrac{\vdots}{\overline{ap_{n-2}}}} \quad \overline{p_{n-1}c}}{\overline{ac}} \quad \dfrac{q_{m-3}q_{m-2} \quad \dfrac{\dfrac{q_3q_4 \quad \dfrac{q_1q_2 \quad \overline{bq_1} \quad \overline{q_2q_3}}{\overline{bq_3}} \quad \overline{q_4q_5}}{\overline{bq_5}}}{\dfrac{\vdots}{\overline{bq_{m-3}}}} \quad \overline{q_{m-2}q_{m-1}}}{\overline{bq_{m-1}}}}{\overline{ab}}$$

Figure A.12

All the NAND-clauses used as axioms in the above proof are on the form $\overline{x_ix_{i+1}}$ and thus elements of NAND_V. All the OR-clauses used as axioms are also on the form x_ix_{i+1}, and since n is odd and m is even, we see that all the OR-clauses used in the proof are indeed from OR_V.

Observe that the case where $b = c$ is unproblematic, since the above proof also proves $\overline{ac}$. The case where some $p_i = q_j$ does not make any of the axioms change, making it too unproblematic.

All the axioms used are thus in Γ_V, which is a subset of Γ_G, and since all the rule applications are correct, our proof is valid.

Bibliography

[1] G. Priest and F. Berto, "Dialetheism," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., Summer 2013, 2013. [Online]. Available: `http://plato.stanford.edu/archives/sum2013/entries/dialetheism/`.

[2] S. Dyrkolbotn and M. Walicki, "Propositional discourse logic," *Synthese*, vol. 191, pp. 863–899, 2014.

[3] M. Bezem, C. Grabmeyer, and M. Walicki, "Expressive power of digraph solvability," *Annals of Pure and Applied Logic*, vol. 163, pp. 200–213, 2012.

[4] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior.* Princeton University Press, 1944 (1947).

[5] V. Chvátal, "On the computational complexity of finding a kernel. technical report crm-300," Centre de Recherches Mathématiques, Univeristé de Montréal, Tech. Rep., 1973.

[6] R. T. Cook, "Patterns of paradox," *The Journal of Symbolic Logic*, vol. 69(3), pp. 767–774, 2004.

[7] M. Richardson, "Solutions of irreflexive relations," *The Annals of Mathematics, Second Series*, vol. 58(3), pp. 573–590, 1953.

[8] S. Yablo, "Paradox without self-reference," *Analysis*, vol. 53(4), pp. 251–252, 1993.

[9] M. Walicki, "Resolving infinitary paradoxes," *The Journal of Symbolic Logic*, [to appear].