Master WIP

Kjetil Midtgarden Golid

May 2, 2016

Contents

1 Introduction **3**
- 1.1 Paradoxes . . . 3
- 1.2 Graph Normal Form . . . 4
- 1.3 Graphs, Kernels and Solutions . . . 5
- 1.4 Discourse Theories and Digraphs . . . 7
- 1.5 Results in Kernel Theory . . . 8
- 1.6 Recognizing dags without kernels . . . 10
- 1.7 Resolving GNF-theories . . . 11
  - 1.7.1 The inference system . . . 12
  - 1.7.2 Inconsistency of the Yablo-graph . . . 13
- 1.8 Thesis overview . . . 14

2 NAND-clauses in graphs **15**
- 2.1 Motivation . . . 15
- 2.2 Odd paths . . . 17
- 2.3 Trimming . . . 19
- 2.4 Odd vels . . . 21
- 2.5 Generalizing trimming . . . 22
- 2.6 Inductive definitions . . . 24
  - 2.6.1 The V_2 construction . . . 24
  - 2.6.2 The V_3 construction . . . 27
  - 2.6.3 V_3 and implication (2) . . . 28
- 2.7 Concluding remarks . . . 31

3 Refutational incompleteness of BNeg **32**
- 3.1 Inconsistencies in general theories . . . 33
  - 3.1.1 Proving the pigeonhole principle . . . 33
  - 3.1.2 Binary-derivability . . . 34
- 3.2 Binary NAND-clauses in graph theories . . . 35
- 3.3 Unary NAND-clauses in graph theories . . . 36
- 3.4 Inconsistencies in graph theories . . . 40

4 Proofs **45**

4.1 Translating CNF to GNF 45
4.2 Inconsistency of Stretched Yablo 47
4.2.1 Proof, D1 and D1b 48
4.2.2 Proof, D2 49
4.2.3 Proof, D3 50
4.2.4 Proof, $\varnothing$ 50
4.3 Provability of NAND-clauses from vels 52
4.4 Axioms in Neg proofs 54

Chapter 1

Introduction

1.1 Paradoxes

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

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

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

A paradox is usually informally defined as something along the lines of *"a statement that can be neither true nor false"*. We can immediately note one thing from this intuitive definition: Since no paradoxes can be true, all paradoxes are, by definition, inconsistent. It is however not the case that all inconsistent theories are paradoxes. Just consider the inconsistent statement, $x \wedge \neg x$ again: this statement simply seems false, and not paradoxical.

A different view is that a paradox is a *dialetheia*, a sentence that is *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 won't change much for our definitions.

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

> "This sentence is false."

If the statement is true, then the statement is false, but if the statement is false, then the statement 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 statements where some of them may refer to themselves or other statements, is called a *discourse* in [2], which we will follow. In order to represent such discourses, we need a formal way of referencing other statements within statements. In propositional logic, this can be done simply by giving statements "names" in the form of adding fresh variables with equivalences to their corresponding statements[1].

Consider the examples below; with normal propositional statements to the left, together with the corresponding *named* statements to the right.

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

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

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

Labelling statements in this way obviously changes their truth value. Even though we have one consistent, one inconsistent and one tautological statement on the left, all the statements become consistent after they have been named. This is because we can find truth values for both x_1, x_2 and x_3 that match 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 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. We are able to label it and then use its label in order to reference itself. Doing this gives us the following result:

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

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

The correspondence between paradoxical discourses and inconsistent sets of labelled statements motivates us to study labelled statements closer, trying to uncover patterns that prevent inconsistencies and thus paradoxes in the represented discourse.

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:

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

[1] TODO: bad wording?

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 Chapter 4.1). Since conjunctive normal form is expressively complete, we get 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 Graphs, Kernels and Solutions

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 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 these two functions formally as follows:

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

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

The number of successors a vertex has is often called the *out-degree* of that vertex.

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 (possibly with the exception of their initial vertices). We let the length of a path be defined by its number of edges.

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

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

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

A kernel is a set of vertices $K \subseteq G$ such that:

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

The above equivalence 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 the existence of kernels 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[2].

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*.

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 node 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 node assigned 1 among its successors. A consequence of this definition is that all sink nodes (nodes 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.

[2]We are concerned with SAT over infinitary formulae in this paper, not the finite version from computer science.

1.4 Discourse Theories and Digraphs

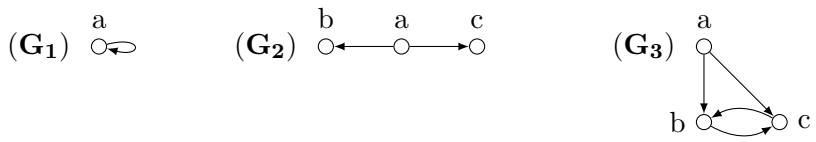
As mentioned earlier, there is a close connection between (1) models of a discourse, (2) kernels of a graph and (3) solutions of a graph. While we have the equivalence between (2) and (3), we will now look at two functions connecting (1) and (2). This correspondence was shown by Roy T. Cook in [6]. We get the following definitions from [3]

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

$\mathcal{G}$: 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 theory $\mathcal{T}(\mathbf{G})$ by taking, for each $x \in G$, the formula $x \leftrightarrow \bigwedge_{y \in N(x)} \neg y$ where $\bigwedge \emptyset = 1$.

Example 1.



The above graphs have the following theories:

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

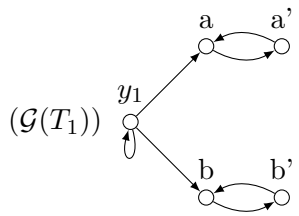
The fact that $sol(\mathbf{G}) = mod(\mathcal{T}(G))$ is shown in [3]. All though not proving it, we can observe that $\mathbf{G_1}$ has no solution, just like its corresponding theory $\mathcal{T}(\mathbf{G_1})$ has no models. $\mathbf{G_2}$ has one solution, where one assigns $a = 0, b = 1, c = 1$. 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 on 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 $x \leftrightarrow \bigwedge_{y \in I_x} y$ make a directed edge $\langle x, y \rangle$ for each $y \in I_x$.

Example 2.

$$(a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (b \leftrightarrow \neg b'), (b' \leftrightarrow \neg b), (y_1 \leftrightarrow (\neg a \wedge \neg b \wedge \neg y_1)) \tag{1.16}$$

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



Again, will we not be proving the correspondence, but notice that T_1 has one model satisfying it, where $a = 0$. $\mathcal{G}(T_1)$ also has one solution, namely where $a = 0, a' = 1, y_1 = 0$. T_2 has three solutions, where either a, b or both are assigned 1. This reflects onto the graph since 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. This equivalence connects logic with graph theory: A graph has a kernel if and only if its corresponding discourse is consistent (non-paradoxical). A theory is paradoxical if and only if its corresponding graph has no kernels. Because of this tight link, we will often refer to graphs without kernels as paradoxical graphs.

The applicability of kernels should by now be obvious. In the next section we will review some of the 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 of the results is the theorem proven by Moses Richardson in 1953:

Theorem 3 ([7])**.** *If D is a finitary*[3] *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.

[3]In a finitary graph, every vertex has a finite number of out-neighbors; the graph has finite branching.

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 $i < j$.

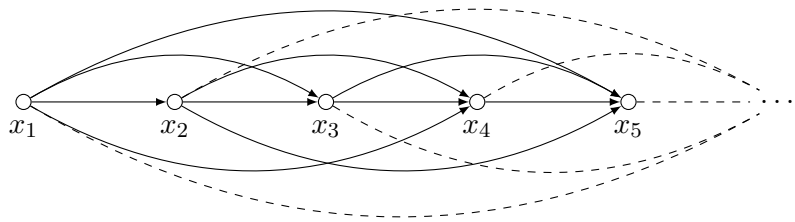


Figure 1.1: The Yablo Graph

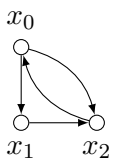
Since there exist no two numbers $x, y \in \mathbb{N}$ such that $x < y$ and $y < x$, we get that the Yablo graph indeed is acyclic[4]. Furthermore, since any natural number has infinitely many numbers strictly larger than it, we get that all the vertices are infinitely branching, making the Yablo graph 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.

Let us assume 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:



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

[4]TODO: This is a bit over-simplified

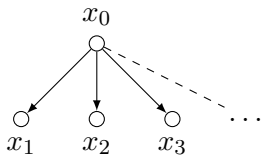


Figure 1.2

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

It is shown in [6] that every digraph (with at least one edge) can be transformed into a infinitary dag such that the assignment α is a solution to the created dag if and only if it is a solution to the original digraph. This means that for any finitary graph that is paradoxical by the virtue of having an odd cycle, there is an 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.

This result will be of great importance to us, enabling us to narrow our search space when looking for paradoxes.

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).

Michał Walicki has proposed the following thesis:

> "If a dag has no kernel then it has a ray with infinitely many vertices dominating it."

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

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.

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, we present an inference system introduced by Walicki in [9] which handles clausal theories induced from GNF-theories.

Recall that a theory written in GNF has formulae of the following form:

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

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

$$x \rightarrow \bigwedge_{i \in I_x} \neg y_i \qquad \text{and} \qquad x \leftarrow \bigwedge_{i \in I_x} \neg y_i \tag{1.18}$$

The first implication can be rewritten in the following way:

$$x \rightarrow \bigwedge_{i \in I_x} \neg y_i \quad \Leftrightarrow \quad \neg x \vee \bigwedge_{i \in I_x} \neg y_i \quad \Leftrightarrow \quad \bigwedge_{i \in I_x} (\neg x \vee \neg y_i) \quad \Leftrightarrow \quad \bigwedge_{i \in I_x} \neg (x \wedge y_i) \tag{1.19}$$

The second implication can be rewritten in the following way:

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

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:

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

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

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 Example 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 mainly 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.

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.

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

$$(1)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{cz}}{\overline{xyz}}abx \qquad (3)\ \frac{\overline{ax} \quad \overline{by}}{\overline{xy}}abx \qquad (2)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{bz}}{\overline{xyz}}abx \tag{1.29}$$

(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.

$$(4)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{cz}}{\overline{xyz}}abc \qquad (5)\ \frac{\overline{ax} \quad \overline{b}}{\overline{x}}ab \qquad (6)\ \frac{\overline{ax} \quad \overline{by} \quad \overline{xyz}}{\overline{xyz}}abx \tag{1.30}$$

The three 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.

We set no restrictions on the number and cardinality of our 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-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 and refutationally complete for theories

with only a countable number of OR-clauses. Soundness gives us that proving $\overline{C}$ for any $C \subseteq G$ implies that the vertices in C cannot all be assigned 1 in the graph model. Refutational completeness gives us that whenever a graph/theory is inconsistent, we are able to prove $\varnothing$ in Neg.

Note that Neg is not generally complete. The following inconsistent graph exemplifies this:

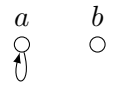


Figure 1.3

Bacause of the loop on vertex a, the graph has no solutions. We therefore have $\vDash \overline{b}$, 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 1.1) 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:

$$\text{NAND} = \{\overline{x_i x_j} \mid i < j\} \qquad \text{OR} = \{x_i x_{i+1} x_{i+2} \ldots \mid i \in \mathbb{N}\} \tag{1.31}$$

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

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

Proving $\varnothing$ is now simple:

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

A less trivial inconsistency proof is the one on the *Stretched Yablo-graph*. This proof can be found in the appendix 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 holds 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 3.1.1.

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 corresponds 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 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 ensures 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 the restrictions makes Neg unable to prove inconsistencies in certain graphs.

Chapter 2

NAND-clauses in graphs

Definition 4. A clause is *unary* if it contains only one atom. A clause is *binary* if it consists of one or two atoms.[1]

In this chapter, we will motivate and conduct the search for graph structural equivalents of binary NAND-clauses proven in Neg. An actual graph structure corresponding to the binary NAND-clauses will not be presented, but we will show some 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:

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

Figure 2.1

The premise will always consist of a collection of NAND-clauses, each of length 1, 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 and formalized by Walicki[9]:

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

[1] TODO: should both lengths be included?

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 the graph structure needed for a kernel not to exist. [2]

The only thing we *do* know about unary NAND-clauses is that they correspond to vertices that are assigned 0 in all models of the graph. We get this from soundness of Neg. This is however not the graph structural property we are ultimately looking for, but we can at least say that 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:

$$\frac{\overline{xy_1} \quad \overline{xy_2} \quad \dots \quad \overline{y_n} \quad \overline{y_{n+1}} \quad \dots}{\overline{x}} \; y_1 y_2 \dots y_n y_{n+1} \dots$$

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, $\overline{xk}$. In addition, 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 $\varnothing$.

We formalize this observation in the following way:

$$\Gamma \vdash \overline{x} \Leftrightarrow \exists K \in \text{OR} : \left(\begin{array}{l} \exists k \in K : G \vdash \overline{kx} \wedge \\ \forall k \in K : G \vdash \overline{kx} \vee G \vdash \overline{k} \end{array} \right) \tag{2.2}$$

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.

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 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 correspondence can now be formally defined as follows:

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

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

[2] TODO: Currently brushing over the restrictions set to the proofs of completeness and soundness. Will fix this later.

The two implications will be referred to as *implication (1)* and *implication (2)*, respectively.

Suppose we manage to find a structure R that satisfied 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.[3]

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

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 the true graph structural equivalent of a binary NAND provable in Neg. We did however 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 where the corresponding NAND-clause is 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 with the figure below, 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.

[3] TODO: This should be rewritten.

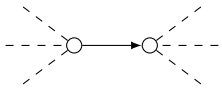


Figure 2.3

The above figure is the basic example of a structure between two vertices that satisfies implication (1). It is however easy to find an example showing how implication (2) does not hold.

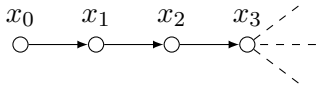


Figure 2.4

The above graph provides us the following axioms: NAND $= \{\overline{x_0x_1}, \overline{x_1x_2}, \overline{x_2x_3}\}$, 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 $\overline{x_0x_3}$:

$$x_1x_2 \quad \dfrac{\overline{x_0x_1} \qquad \overline{x_2x_3}}{\overline{x_0x_3}}$$

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:

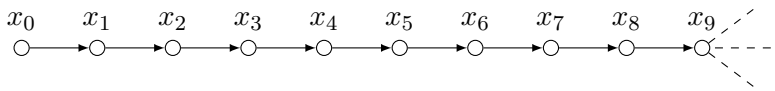


Figure 2.6

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

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.

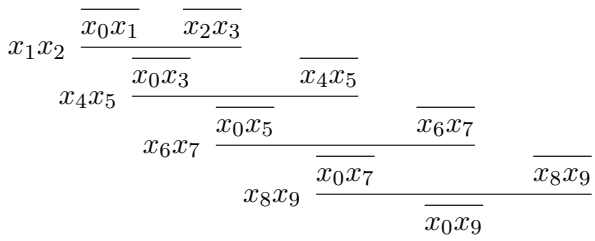


Figure 2.7

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 $\overline{ab}$ is unprovable in Neg. The kernel in the graph below is represented by the black vertices.

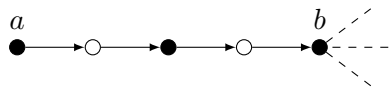


Figure 2.8

Restricting our paths to be of odd length lets us avoid cases like the one above, but there are still some cases left to avoid, in order for our implication to hold.

2.3 Trimming

We introduce the following terminology:

Definition 5. 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 $N(x) = \{y\}$.

Definition 6. 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.

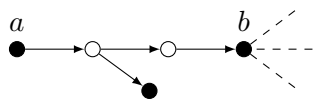


Figure 2.9

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

$$\dfrac{x_{k-2}x_{k-1} \quad \dfrac{\dfrac{x_3x_4 \quad \dfrac{\overline{x_0x_1} \quad \overline{x_2x_3}}{\overline{x_0x_3}} \quad \overline{x_4x_5}}{\overline{x_0x_5}} \quad \vdots}{\overline{x_0x_{k-2}}} \quad \overline{x_{k-1}x_k}}{\overline{x_0x_k}}$$

Figure 2.10

All the OR-clauses used are indeed 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:

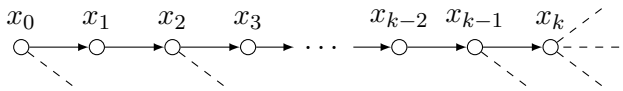


Figure 2.11: An oddly trimmed path of odd length

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 7. A path $\langle x_0, x_k \rangle$[4] 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 above proof, 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 are vel-connected, thus being a graph structure satisfying implication (1).

[4] TODO: This notation is not explained

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:

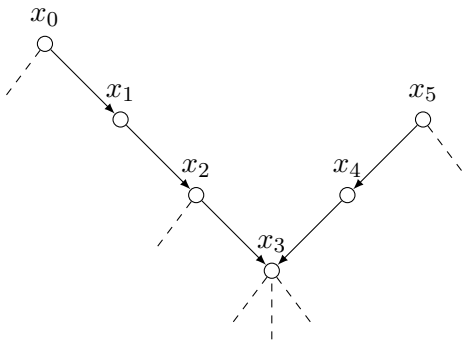


Figure 2.12

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 8. 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 Section 4.3.

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.

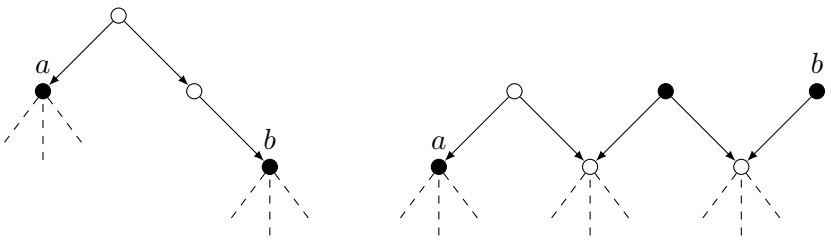


Figure 2.13

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 of these constructions are providing 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 Figure 2.9. The figure shows how a path that normally provides a provable NAND-clause may be "ruined" by adding out-edges to certain vertices in the path, making certain vital OR-clauses non-binary and thus making the NAND-clause unprovable.

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

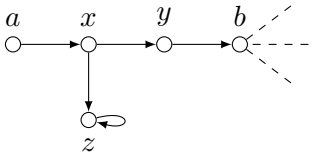


Figure 2.14

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 are provably false in Neg. 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 notion of trimmed vertices.

The big problem with this generalization is that it makes our vel-relation no longer a purely graph-structural one. 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. This is exactly what we try to avoid with our vel-relation.

To make matters worse, consider the following two graphs:

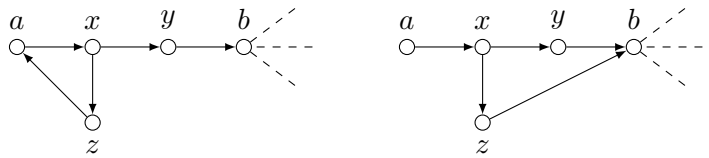


Figure 2.16

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.

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

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 $\overline{z}$, $\overline{az}$ or $\overline{bz}$ are provable in Neg.

This means that our concept of trimming needs further weakening, taking into account the observation made above. Such a weakening will simply add the case where, given a vel between a and b, trimmed vertices may branch off to vertices c as long as either $\overline{ac}$ or $\overline{bc}$ are provable in Neg.

The definition now bases the provability of $\overline{ab}$ on the provability of a set of other binary NAND-clauses, suggesting an inductive approach for our definitions.

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 8) 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.

$$\textbf{(BC):} \quad N(a,b) \vee N(b,a) \qquad \Rightarrow V_1(a,b) \tag{2.10}$$

$$\textbf{(IS):} \quad \exists c \in N(a) : (N(c) = \{d\} \wedge V_1(d,b)) \qquad \Rightarrow V_1(a,b) \tag{2.11}$$

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 construction

It is now much easier to formally express the new, weaker concept of trimming. Let $V_2 \subseteq G \times G$ be the set of all pairs of vertices related by our new vel-definition. V_2 can be defined inductively in the following way, again with a and b as arbitrary vertices from G.

$$\textbf{(BC):} \quad N(a,b) \vee N(b,a) \qquad \Rightarrow V_2(a,b) \tag{2.12}$$

$$\textbf{(IS):} \quad \exists c \in N(a) : \left(\begin{array}{l} \exists d \in N(c) : V_2(d,b) \quad \wedge \\ \forall d \in N(c) : V_2(d,b) \vee V_2(d,a) \vee V_2(d,d) \end{array} \right) \Rightarrow V_2(a,b) \tag{2.13}$$

Comparing the two inductive definitions, it is easy to see that V_2 is a generalization of V_1.

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 $\overline{ab}$ will be an axiom and thus trivially provable.

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

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

Figure 2.18

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

$$d \in D_b \Leftrightarrow V_2(d,b), \quad d \in D_a \Leftrightarrow V_2(d,a), \quad d \in D_d \Leftrightarrow V_2(d,d) \tag{2.14}$$

The current situation is illustrated in Figure 2.19

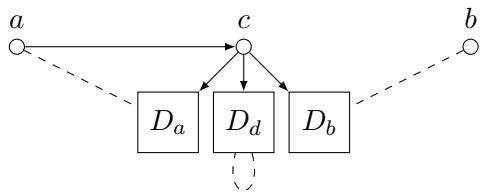


Figure 2.19: The inductive case of a V_2 construction. Boxes represents sets of vertices. An edge between a vertex and a box represents a set of edges from the vertex to each of the vertices in the set.

The inductive part of the V_2 definition (2.13) 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 2.18 results the following proof:

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

Figure 2.20

The nonemptiness of D_b gives us the existence of a b in the premise, while he 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 combined 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).

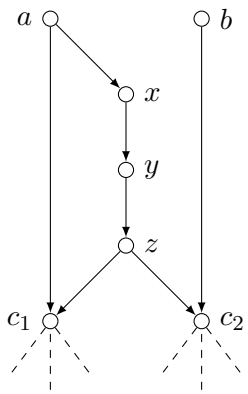


Figure 2.21

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

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

Figure 2.22

To show that the graph in Figure 2.21 is not an instance of V_2 demands a closer look at the graph.

Assume $V_2(a, b)$. 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.

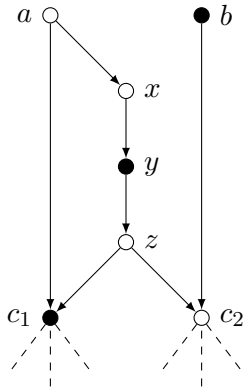


Figure 2.23

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 construction

Now, in what way can V_2 be generalized in order to include the graph in Figure 2.21? Looking at the proof in Figure 2.20, 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 construction V_3 which is a generalization of V_2. Like in the definitions of V_1 and V_2, a and b are vertices from G.

$$\textbf{(BC):} \quad N(a,b) \vee N(b,a) \qquad \Rightarrow V_3(a,b) \tag{2.15}$$

$$\textbf{(IS):} \quad \exists c \in G : \left(\begin{array}{ll} \exists d \in N(c) \cup \{c\} : V_3(d,a) & \wedge \\ \exists d \in N(c) \cup \{c\} : V_3(d,b) & \wedge \\ \forall d \in N(c) \cup \{c\} : V_3(d,a) \vee V_3(d,b) \vee V_3(d,d) & \end{array} \right) \quad \Rightarrow V_3(a,b) \tag{2.16}$$

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 2.21, 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 2.24

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

□

2.6.3 V_3 and implication (2)

Just like with V_2, if V_3 also satisfies implication (2), it has some considerable consequences for our proof system, Neg. Figure 2.24 shows that if two vertices a and b are V_3-related, not only is $\overline{ab}$ provable in Neg, it is provable in such a way that the premise of the last rule application contains binary NAND-clauses only (the proof has a *binary root*).

If V_3 satisfies implication (1) and (2), then for any provable binary NAND-clause $\overline{ab}$, by implication (1) and (2), it has a proof with a binary root. Applying now the same reasoning on all of the binary NAND-clauses in its premise, we get that all of these also have proofs with binary roots. Doing this repeatedly throughout the whole proof of $\overline{ab}$ ultimately gives us a proof containing exclusively binary NAND-clauses. Implication (2) thus gives us that for any provable binary NAND-clause there exists a proof in Neg with binary NAND-clauses only. We will however see that implication (2) does *not* hold for V_3.

The graph in Figure 2.25 provides a provable NAND-clause $\overline{ab}$, but is not an instance of V_3.

The graph is admittedly a bit over-simplified. Even though the vertices y_1, y_2 and c_2 are depicted like sinks, we will be treating 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.

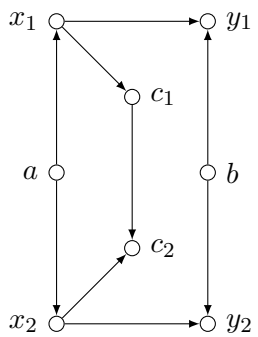


Figure 2.25

Here is one way to prove $\overline{ab}$ in Neg given the graph theory from Figure 2.25:

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

Figure 2.26

Now, 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 can be shown to be impossible.

Consider the following six solutions of the graph:

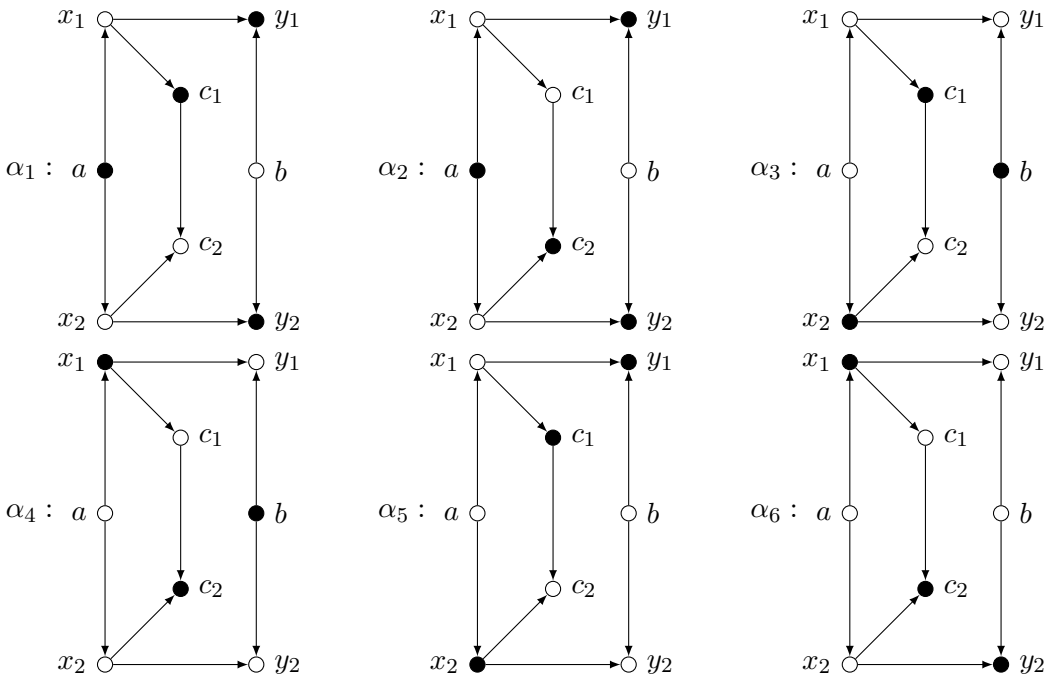


Figure 2.27

Given any pair of vertices x, y, if there exists an assignment 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 2.28 shows what pairs of vertices can be 1 under the same assignment 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.

	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 2.28

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 thus left with the following collection of 11 "not-unprovable" binary NAND-clauses:

$$\{\ \overline{ab},\ \overline{ax_1},\ \overline{ax_2},\ \overline{by_1},\ \overline{by_2},\ \overline{x_1x_2},\ \overline{x_1y_1},\ \overline{x_1c_1},\ \overline{x_2y_2},\ \overline{x_2c_2}\ \overline{c_1c_2}\ \} \tag{2.17}$$

By the definition of V_3 from (2.12) and (2.13), in order for a and b to be V_3-related, a vertex c 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.

It turns out that none of the vertices in the graph satisfies this requirement. a and b are therefore not V_3-related, and 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 V, independent of the definition of Neg, such that $V(a,b) \Leftrightarrow \vdash \overline{ab}$ (implication (1) and (2)). This would as a result 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 2.25). 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 9. Let *BNeg* denote the proof system Neg when restricted to using binary NAND-clauses only[1]

Definition 10. A clause is *binary-derivable* if it is provable in BNeg.

This chapter will investigate the following conjecture:

Conjecture 11. *BNeg is refutationally complete.*

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 unary and 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 provable binary NAND-clauses are not binary-derivable. Section 3.3 will build on this result to show that there exist provable unary NAND-clauses that are not binary-derivable. Lastly, Section 3.4 will present the final proof showing an inconsistency that is not binary-derivable, disproving Conjecture 11.

[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) all n pigeons are 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:

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

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

3.1.1 Proving the pigeonhole principle

The pigeonhole principle for 4 pigeons and 3 holes will have the following axioms:

$$\text{OR} = \{\ 1_1 2_1 3_1,\ 1_2 2_2 3_2,\ 1_3 2_3 3_3,\ 1_4 2_4 3_4\ \} \tag{3.3}$$

$$\text{NAND} = \left\{ \begin{array}{l} \overline{1_1 1_2},\ \overline{1_1 1_3},\ \overline{1_1 1_4},\ \overline{1_2 1_3},\ \overline{1_2 1_4},\ \overline{1_3 1_4}, \\ \overline{2_1 2_2},\ \overline{2_1 2_3},\ \overline{2_1 2_4},\ \overline{2_2 2_3},\ \overline{2_2 2_4},\ \overline{2_3 2_4}, \\ \overline{3_1 2_2},\ \overline{3_1 3_3},\ \overline{3_1 3_4},\ \overline{3_2 3_3},\ \overline{3_2 3_4},\ \overline{3_3 3_4} \end{array} \right\} \tag{3.4}$$

From the axioms above we can prove inconsistency in the following way:

Proving $\overline{1_4}$:

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

Proving $\overline{2_4}$:

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

Proving $\overline{3_4}$:

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

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

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

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 and 3.4

3.1.2 Binary-derivability

As observed in 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. 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.

$$1_i 2_i 3_i \quad \dfrac{\overline{1_i 1_j} \quad \overline{2_i 2_k} \quad \overline{3_i 3_l}}{\overline{1_j 2_k 3_l}}$$

Figure 3.1

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, as long as the OR-clauses are larger than 2, 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, it is the graph theories that motivated the conjecture in the first place.

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

The conjecture will be disproven 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. We will then make an attempt to extend the graph, making it inconsistent, and then show that its inconsistency proof has to include the non-binary-derivable NAND-clause. This attempt will however prove to be unsuccessful.

Let us again consider the graph from Figure 2.25, Section 2.6.3:

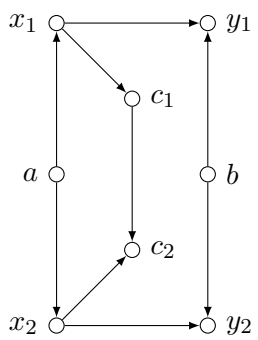


Figure 3.2

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

The NAND-clause we will show not to be binary-provable is $\overline{ab}$. Figure 2.26 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 2.28 worked out in Section 2.6.3. 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 3.3

Let us assume that we have 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, one containing a y and one containing a c. Looking at our table again, we see that the potentially provable NAND-clauses containing a c 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, thus contradicting our assumption. We can therefore conclude that $\overline{ab}$ is *not* binary-derivable, thus disproving Conjecture 12.

3.3 Unary NAND-clauses in graph theories

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

We use our graph from Figure 3.2 to make the following, bigger graph:

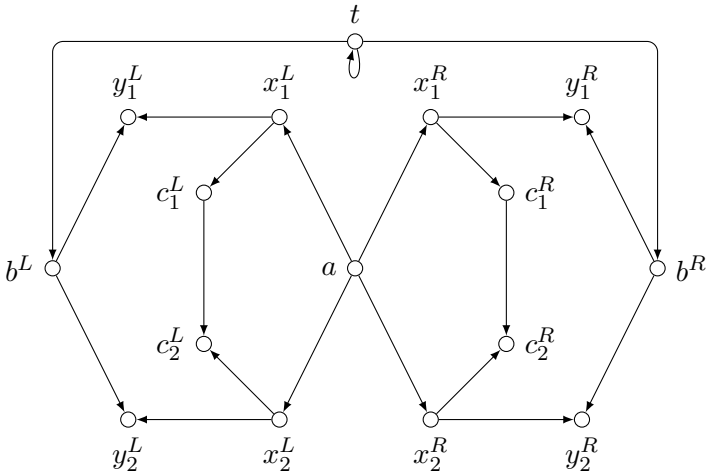


Figure 3.4

The above graph contains two copies of the graph from Figure 3.2, 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 2.26. This makes the proof of $\overline{a}$ trivial:

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

Figure 3.5

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

Lemma 13. *Based on the graph in Figure 3.4, 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.

$$
\dfrac{\overline{ax_1^L} \quad \dfrac{b^L y_1^L y_2^L \quad \dfrac{\dfrac{tb^R b^L \quad \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^L y_1^L}}{\overline{ty_1^L}} \quad \overline{tb^L} \quad \overline{x_2^L y_2^L}}{\overline{tx_2^L}} \quad \overline{c_1^L c_2^L} \quad \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^L y_2^L}}{\overline{ty_2^L}}}{x_2^L y_2^L c_2^L} \quad tb^R b^L}{\overline{tc_1^L}} \quad \dfrac{\bar{t} \quad \overline{tb^R} \quad \overline{b^L y_1^L}}{\overline{ty_1^L}} \quad tb^R b^L}{\overline{at}} \quad x_1^L y_1^L c_1^L
$$

Figure 3.6

We now show that $\overline{at}$ is the only one. This is done by structural induction on the complexity of the proof of 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 length is 1, so all the clauses in the premise must be axiomatic. That includes at least one clause containing a and at least one clause containing γ. Figure 3.7 illustrates the situation.

$$
\dfrac{\overline{ax_i^K} \quad \overline{\gamma y_i^K} \quad \overline{\gamma c_i^K}}{\overline{a\gamma}} \quad x_i^K y_i^K c_i^K
$$

Figure 3.7

The only axiomatic NAND-clauses containing a are on the form $\overline{ax_i^K}$ (K being either L or R and i either 1 or 2). The OR-clause used must therefore be the corresponding $x_i^K y_i^K c_i^K$, being the only OR-clause containing x_i^K. Since the OR-clause is of length 3, the premise must contain two other clauses and they must have the following form in order to give us $\overline{a\gamma}$ in the conclusion: $\overline{\gamma y_i^K}, \overline{\gamma c_i^K}$. The only pairs of axioms that matches this description are the ones where $\gamma = x_i^K$. With a proof length of 1, we are thus not able to prove binary NAND-clauses containing a that are non-axiomatic.

Suppose we have a proof that concludes with some binary NAND-clause $\overline{a\gamma}$; like in the base case, 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. The induction hypothesis gives us that if the NAND-clause containing a is non-axiomatic, then it must be $\overline{at}$. All NAND-clauses containing a in the premise must therefore be either $\overline{at}$ ore one of the four axioms on the form $\overline{ax_i^K}$.

Let us first consider the case where some $\overline{ax_i^K}$ is in the premise. This case is similar to the base case. Like in the base case, the only applicable OR-clause is the corresponding $x_i^K y_i^K c_i^K$, so the premise contains 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. Neither $\overline{y_i^K}$ nor $\overline{c_i^K}$ is

provable[2], so they both have to contain γ. Thus, like in the base case, 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[3] γ-substitution that makes both of these NAND-clauses provable is $\gamma = t$, giving us $\overline{at}$ in the conclusion.

The other case is where $\overline{at}$ appears in the premise. Figure 3.8 illustrates this situation.

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

Figure 3.8

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. None of them are provably false[4], and none of them can contain a because of induction hypothesis, so they both must contain some γ. The only[5] 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 $\overline{at}$. □

It is now easy to show that $\overline{a}$ is indeed not binary-provable. 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 the above proof, 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.

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

[2] TODO: show assignments
[3] TODO: show assignments
[4] TODO: show assignments
[5] TODO: show assignments

3.4 Inconsistencies in graph theories

In this section, we will prove the existence of non-binary-derivable inconsistencies from graphs. We will first prove a lemma, then apply this lemma to an extended version of the graph from Figure 3.4 to show that its inconsistency is not binary-derivable.

Let G be a digraph with a source vertex (a vertex with no predecessors) x. Let A and B be two disjoint subgraphs of G such that $N(A) \subseteq A$ and $N(B) \subseteq B$ and such that $N^-(A) \setminus A = N^-(B) \setminus B = \{x\}$. In words, nothing points out of A and B and only x points in. Note that x might have additional successors.

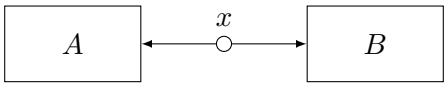


Figure 3.9: Graph G

A couple of observations can be made from the above construction:

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

Based on graph G from Figure 3.9 we now introduce the following lemma:

Lemma 14. *Given two vertices $a \in A$ and $b \in B$; if $\overline{ab}$ is provable in Neg, then the proof must contain the OR-clause X.*

The lemma will be proven by inducing over the length of the proof of $\overline{ab}$.

Proof. In the base case, the proof length is 1, so $\overline{ab}$ is the direct conclusion from a premise with axioms only. The premise must contain one axiom on the form $\overline{ap}$ and one

$$\frac{\overline{ap} \quad \overline{bq} \quad \dots}{\overline{ab}} \; X$$

Figure 3.10

on the form $\overline{bq}$. If either $p = x$ or $q = x$, then the OR-clause used in the proof step must be X, being the only OR-clause containing x. On the other hand, if neither p nor q equals x, then $p \in A$ and $q \in B$. Since X is the only OR-clause containing vertices from both A and B, X is again our only choice. A proof of $\overline{ab}$ of length 1 must hence use the OR-clause X.

For the inductive step, we assume that for any proof of length $k-1$; if it concludes with a NAND-clause $\overline{pq}$ such that p and q are from different components, then it must use the OR-clause X somewhere. This assumption makes us able to prove the inductive step in almost exactly the same way as with the base case.

Suppose there is a proof of length k concluding with $\overline{ab}$. Its immediate premise must contain one NAND-clause on the form $\overline{ap}$ and one on the form $\overline{bq}$. Again, if either $p = x$ or $q = x$, or if $p \in A$ and $q \in B$, the OR-clause X must be used, by the same argument as in the base case. This leaves us with the case where either $p \notin A$ or $q \notin B$. Our induction hypothesis gives us that the proof of either of those contains X somewhere.

A proof of length k concluding with $\overline{ab}$ must therefore use the OR-clause X, so we can conclude that *any* proof of $\overline{ab}$ must contain X. $\square$

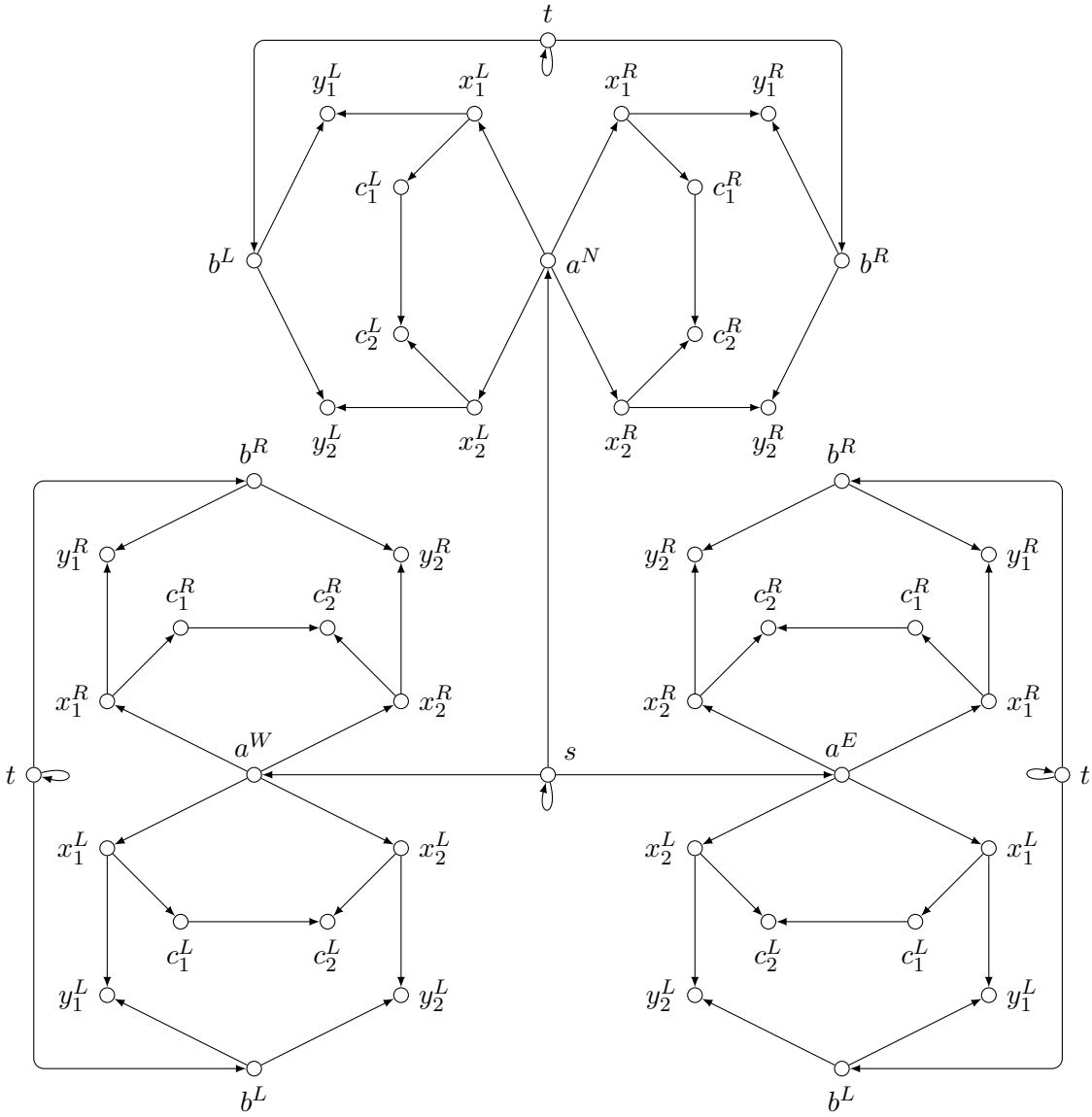


Figure 3.11

Consider now the graph in Figure 3.11 containing 3 copies of our graph from Figure 3.4; one northern (N), one western (W) and one eastern (E).

The proof of $\overline{a}$ from Figure 3.5 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 3.12. The graph is thus provably inconsistent. We will now show that the inconsistency is non-binary-derivable.

Proof. The first thing to notice is that if one removes the loop on vertex s, the graph is no

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

Figure 3.12

longer inconsistent. This tells us that the NAND-clause $\overline{s}$ must be in any inconsistency proof. If that was not the case, we would be able to prove $\varnothing$ on a graph that was not inconsistent, violating the fact that Neg is sound.

Having that any inconsistency proof contains $\overline{s}$ immediately gives us that any inconsistency proof contains the OR-clause $sa^N a^W a^E$, since this is the only OR-clause containing s. We thus have the following situation somewhere in the proof, where the sets of vertices P, Q, R might be empty:

$$\dfrac{\overline{s} \qquad \dfrac{\cdots}{\overline{a^N P}} \qquad \dfrac{\cdots}{\overline{a^W Q}} \qquad \dfrac{\cdots}{\overline{a^W R}}}{\overline{P \cup Q \cup R}}\ sa^N a^W a^E$$

Figure 3.13

The OR-clause in question might obviously appear several places in the proof, but let us assume that this is its *first* appearance[6], i.e. none of the NAND-clauses in the premise are proven using the OR-clause $sa^N a^W a^E$.

First of all, neither of the sets P, Q, R can contain more than 1 element, since that would make the NAND-clauses non-binary. For the same reason can their union never contain more than two elements, since that would make the concluding NAND-clause non-binary. One of them must thus either be empty, or some of them must contain the same element.

Let us first assume that neither of P, Q, R are empty. This implies that at least two of them contain the same element. We will now apply Lemma 14 to show that this is not possible.

Observe that our graph consists of three isolated components, N, W and E, only connected by the vertex s having a vertex from each component in its neighborhood. Let, without any loss of generality, P and Q be the two sets with the same element, i.e., $P = Q = \{p\}$. That gives us the two NAND-clauses $\overline{a^N p}$ and $\overline{a^W p}$. Since our three components are mutually disjoint, at least one of the above NAND-clauses must contain vertices from two different components.[7] Lemma 14 gives us that the proof of that NAND-clause therefore must contain the OR-clause $sa^N a^W a^E$, which contradicts our assumption of it being used for the first time in Figure 3.13. Therefore, none of the three

[6] TODO: the first appearance of the OR-clause might not have a unary $\overline{s}$ in the premise. this should be mentioned. It does however not change our argumentation

[7] TODO: one of them can be s; should be mentioned.

a-clauses in the premise can contain the same element, leaving us with the case where one of them is unary, i.e. either P, Q or R is empty.

Now, we know that both $\overline{a^N}$, $\overline{a^W}$ and $\overline{a^E}$ are provable, but the only proof we have seen has been non-binary, using the proofs of $\overline{ab}$ as subproofs. The last section showed that $\overline{a}$ was not provable if one was only to use the vertices from within its own component. So if one of the a-clauses in the premise in Figure 3.13 is unary, then it has to have been proved using either s or vertices from other components. In both cases, we get that such a proof must contain the OR-clause $sa^N a^W a^E$ somewhere, again contradicting our assumption.

All in all, this gives us that if the proof is binary up until this rule application, the concluding NAND-clause cannot be binary. Since the rule application has to be present in the inconsistency proof, the proof therefore has to contain a non-binary NAND-clause, indeed making the inconsistency non-binary-derivable. □

While Neg in general is not complete, this proof shows that when restricted to binary NAND-clauses, Neg is not even refutationally complete, being unable to recognize certain inconsistencies.

Chapter 4

Proofs

4.1 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' fom 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 some examples of CNF-theories with their corresponding GNF-theories.

Example 15.

$$\text{CNF:} \quad (a \vee b) \tag{4.1}$$

$$\text{GNF:} \quad (a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (b \leftrightarrow \neg b'), (b' \leftrightarrow \neg b), (y_1 \leftrightarrow (\neg a \wedge \neg b \wedge \neg y_1)) \tag{4.2}$$

$$\tag{4.3}$$

$$\text{CNF:} \quad (\neg a) \tag{4.4}$$

$$\text{GNF:} \quad (a \leftrightarrow \neg a'), (a' \leftrightarrow \neg a), (y_1 \leftrightarrow (\neg a' \wedge \neg y_1)) \tag{4.5}$$

$$\tag{4.6}$$

4.2 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 $(2 \times (y - x) - 1)$.

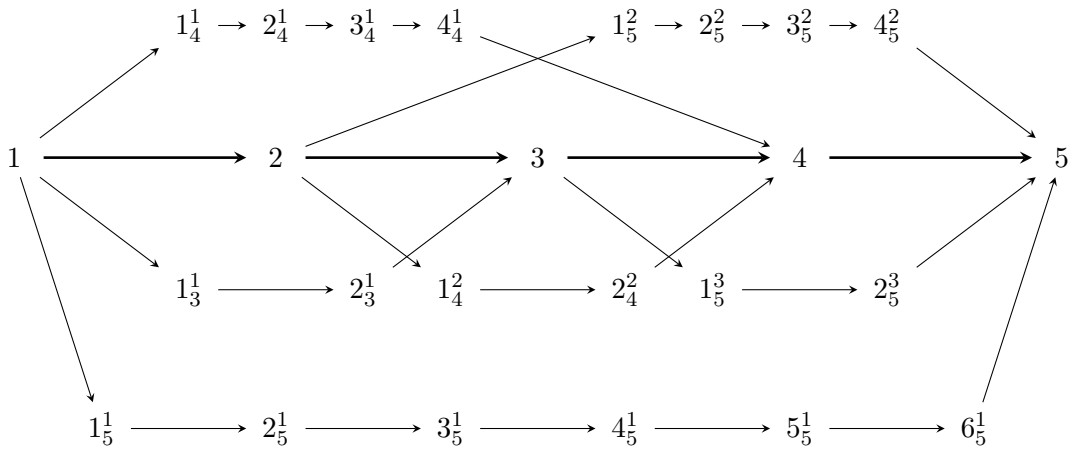


Figure 4.1: Stretched Yablo

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_y^x 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 $L_y^x = 2(y-x)-1$ (from the definition above), each path consists of $L_y^x - 1 = 2(y-x)-2$ peripheral vertices. The vertex pointing back onto the core ray will thus alway be the vertex $(2(y-x)-2)_y^x$.

Based on the given definition of Stretched Yablo and the notation from above, we get

the following sets of axioms:

$$N1a = \{ \overline{x(x+1)} \mid x \in \mathbb{N} \} \tag{4.7}$$
$$N1b = \{ \overline{x1^x_y} \mid x, y \in \mathbb{N}, x+1 < y \} \tag{4.8}$$
$$N2 = \{ \overline{n^x_y(n+1)^x_y} \mid n, x, y \in \mathbb{N}, x+1 < y, n < 2(y-x)-2 \} \tag{4.9}$$
$$N3 = \{ \overline{n^x_y y} \mid n, x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2 \} \tag{4.10}$$
$$O1 = \{ x(x+1)1^x_{x+2}1^x_{x+3}\dots \mid x \in \mathbb{N} \} \tag{4.11}$$
$$O2 = \{ n^x_y(n+1)^x_y \mid n, x, y \in \mathbb{N}, x+1 < y, n < 2(y-x)-2 \} \tag{4.12}$$
$$O3 = \{ (n^x_y y \mid n, x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2 \} \tag{4.13}$$

$$\text{NAND} = N1a \cup N1b \cup N2 \cup N3 \qquad \text{OR} = O1 \cup O2 \cup O3 \tag{4.14}$$

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:

$$D1 = \{ \overline{xy} \mid x, y \in \mathbb{N}, x \neq y \} \tag{4.15}$$
$$D1b = \{ \overline{1^x_y n^x_y} \mid x, y \in \mathbb{N}, x+1 < y, n = 2(y-x)-2 \} \tag{4.16}$$
$$D2 = \{ \overline{x1^y_z} \mid x, y, z \in \mathbb{N}, x \neq z, y+1 < z \} \tag{4.17}$$
$$D3 = \{ \overline{1^1_x 1^y_z} \mid x, y, z \in \mathbb{N}, x \neq z, y+1 < z \} \tag{4.18}$$

4.2.1 Proof, D1 and D1b

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

$$
\begin{array}{rcccc}
 & \overset{(N1b)}{\overline{x1^x_y}} & \overset{(N2)}{\overline{2^x_y3^x_y}} & & \\
\text{(O2) } 1^x_y2^x_y & \multicolumn{2}{c}{\overline{x3^x_y}} & \overset{(N2)}{\overline{4^x_y5^x_y}} & \\
\text{(O2) } 3^x_y4^x_y & \multicolumn{3}{c}{\overline{x5^x_y}} & \overset{(N2)}{\overline{6^x_y7^x_y}} \\
\text{(O2) } 5^x_y6^x_y & \multicolumn{4}{c}{\overline{x7^x_y}} & \\
 & \multicolumn{4}{c}{\vdots} & \\
\text{(O2) } (n-1)^x_yn^x_y & \multicolumn{3}{c}{\overline{x(n-1)^x_y}} & \overset{(N3)}{\overline{n^x_yy}} \\
 & \multicolumn{4}{c}{\overline{xy}}
\end{array}
$$

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:

$$
\begin{array}{rcccc}
 & \overset{(N2)}{\overline{1^x_y2^x_y}} & \overset{(N2)}{\overline{3^x_y4^x_y}} & & \\
\text{(O2) } 2^x_y3^x_y & \multicolumn{2}{c}{\overline{1^x_y4^x_y}} & \overset{(N2)}{\overline{5^x_y6^x_y}} & \\
\text{(O2) } 4^x_y5^x_y & \multicolumn{3}{c}{\overline{1^x_y6^x_y}} & \overset{(N2)}{\overline{7^x_y8^x_y}} \\
\text{(O2) } 6^x_y7^x_y & \multicolumn{4}{c}{\overline{1^x_y8^x_y}} & \\
 & \multicolumn{4}{c}{\vdots} & \\
\text{(O2) } (n-2)^x_y(n-1)^x_y & \multicolumn{3}{c}{\overline{1^x_y(n-2)^x_y}} & \overset{(N2)}{\overline{(n-1)^x_yn^x_y}} \\
 & \multicolumn{4}{c}{\overline{1^x_yn^x_y}}
\end{array}
$$

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

4.2.2 Proof, D2

Again, we 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:

$$
\begin{array}{rcc}
 & \overset{(D1)}{\overline{xz}} & \overset{(D1b)}{\overline{1^y_zn^y_z}} \\
\text{(O3) } n^y_zz & \multicolumn{2}{c}{\overline{x1^y_z}}
\end{array}
$$

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.

4.2.3 Proof, D3

n now denotes the number of vertices in the peripheral path between y and z.

$$\text{(O3)}\ n_z^y z \quad \dfrac{\dfrac{(D2)}{\overline{1_x^1 z}} \qquad \dfrac{(D1b)}{\overline{1_z^y n_z^y}}}{\overline{1_x^1 1_z^y}}$$

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

4.2.4 Proof, $\varnothing$

We will prove the inconsistency of Stretched Yablo by first proving $\overline{1}, \overline{2}, \overline{1_3^1}, \overline{1_4^1 1_5^1} \ldots$.

Proving $\overline{1}$:

$$\text{(O1)}\ 231_4^2 1_5^2 1_6^2 \ldots \quad \dfrac{\dfrac{(N1)}{\overline{12}} \quad \dfrac{(D1)}{\overline{13}} \quad \dfrac{(D2)}{\overline{11_4^2}} \quad \dfrac{(D2)}{\overline{11_5^2}} \quad \dfrac{(D2)}{\overline{11_6^2}} \quad \ldots}{\overline{1}}$$

Proving $\overline{2}$:

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

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

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

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

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

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

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

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

4.3 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 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.

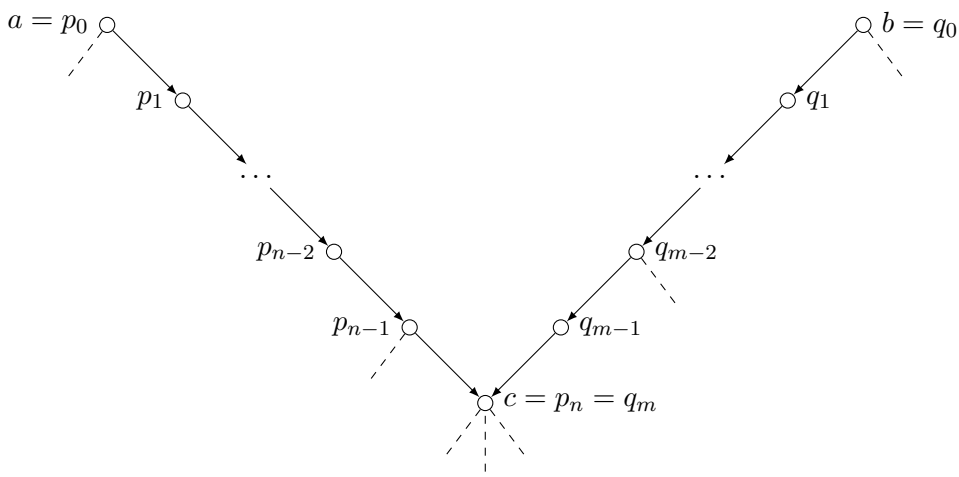


Figure 4.2: A general vel between a and b

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:

$$\text{NAND}_V = \{\ \overline{p_i p_{i+1}} \mid 0 \leq i < n\ \} \cup \{\ \overline{q_j q_{j+1}} \mid 0 \leq j < m\ \} \tag{4.19}$$

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:

$$\mathrm{OR}_V = \{\ p_i p_{i+1} \mid 0 \leq i < n, i \text{ is odd}\ \} \cup \{\ q_j q_{j+1} \mid 0 \leq j < m, j \text{ is odd}\ \} \tag{4.20}$$

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

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

Figure 4.3: Proof that a and b are vel-connected

All the NAND-clauses used as axioms in the above proof are on the form $\overline{x_i x_{i+1}}$ and thus elements of NAND_V. All the OR-clauses used as axioms are also on the form $x_i x_{i+1}$, 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.

4.4 Axioms in Neg proofs

Definition 16. A Neg proof is *thin* if each rule application in the proof contains an axiomatic NAND-clause in its premise.

In this section, we will prove the following statement:

Theorem 17. *Any NAND-clause provable in Neg has a thin proof.*

Equivalently, for any Neg-proof P there exists a thin Neg-proof with the same conclusion. We will prove this theorem by inducing over the height of P.

Base case: If P is of height 1, i.e. it consists only of one rule application, then it is trivially thin, since all the NAND-clauses in the premise must be axioms.

Inductive step: The induction hypothesis lets us assume that for any proof with a height of $k-1$ or less, there exists a thin proof with the same conclusion. We will use this hypothesis to prove the same thing for proofs of height k. Given a proof with a height of k, if the kth rule application contains an axiom, we are done. Therefore, suppose that the kth rule application does *not* contain an axiom.

The situation will look like this:

$$(\text{height}{:}k)\ \dfrac{\dfrac{\cdots}{\overline{a_1A_1}}\,r_1 \qquad \dfrac{\cdots}{\overline{a_2A_2}}\,r_2 \qquad \dfrac{\cdots}{\overline{a_3A_3}}\,r_3 \qquad \cdots \qquad \dfrac{\cdots}{\overline{a_nA_n}}\,r_n}{\bigcup_{i\leq n}\overline{A_i}}\ a_1a_2a_3\ldots a_n$$

Figure 4.4

We will show that the above proof can be *transformed* into a thin proof, all while preserving the subproofs and the conclusion.

For each rule application r_i in Figure 4.4, letting $r_i = b_1^ib_2^i\ldots b_m^i$, we can write its premise as the following collection of NAND-clauses: $\overline{b_1^iB_1^i}$, $\overline{b_2^iB_2^i}$, $\ldots$, $\overline{b_j^ia_iB_j^i}$, $\overline{b_m^iB_m^i}$ such that $\bigcup_{j\leq m}B_j^i = A_i$. Note that the NAND-clause with the index j contains a_i. We will denote this index as j_i.

Based purely on these facts, each rule application r_i can be represented as follows without any loss of generality:

$$(\text{height}{:}k-1)\ \dfrac{\dfrac{\cdots}{\overline{b_1^iB_1^i}} \qquad \dfrac{\cdots}{\overline{b_2^iB_2^i}} \qquad \cdots \qquad \dfrac{\cdots}{\overline{b_{j_i}^ia_iB_{j_i}^i}} \qquad \cdots \qquad \dfrac{\cdots}{\overline{b_m^iB_m^i}}}{\overline{a_iA_i}}\ b_1^ib_2^i\ldots b_m^i$$

Figure 4.5

Now, since the premise of each rule application proving $\overline{a_iA_i}$ has a NAND-clause containing a_i, we can make the following proof:

$$\dfrac{\dfrac{\dfrac{\dots}{\overline{b^1_{j_1} a_1 B^1_{j_1}}} \quad \dfrac{\dots}{\overline{b^2_{j_2} a_2 B^2_{j_2}}} \quad \dfrac{\dots}{\overline{b^3_{j_3} a_3 B^3_{j_3}}} \quad \dots \quad \dfrac{\dots}{\overline{b^n_{j_n} a_n B^n_{j_n}}}}{\overline{\bigcup_{i \leq n} b^i_{j_i} B^i_{j_i}}}}{a_1 a_2 a_3 \dots a_n}$$

Figure 4.6

Since we defined the proof in Figure 4.4 to be of height k, all the NAND-clauses on the form $a_i Ai$ from that proof can itself not have a proof higher than $k-1$. For the same reason, every NAND-clause on the form $b_i B_i$ from Figure 4.5 has height of at most $k-2$.

The proof in Figure 4.6 therefore has a height of at most $k-1$, so by our induction hypothesis there exists a thin proof with the same conclusion.

For tomorrow: Use minimum height for tree, not maximum height. In other words, induce over the length of the path to the nearest axiom.

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].