UNIVERSITY OF OSLO
Department of Informatics

Instance-Based Hyper-Tableaux for Coherent Logic

Master Thesis

Evgenij Thorstensen

May 2009

Contents

Acknowledgments

> Silent gratitude isn't much use
> to anyone.
>
> G.B. Stern

First and foremost, I wish to thank my advisor **Martin Giese**. Working with him has been interesting and fun, and when I was stuck he always had time to sit down with me and help me stare at the proof. Martin knows virtually everything and shares his knowledge freely, explaining the difficult things clearly and patiently. He also reads with great care; this thesis would be far more nebulous without his critical look. Martin, your dedication to your work has inspired me to work harder and think better — thank you.

I also wish to thank my second advisor **Arild Waaler**, who has a gift for saying innocent things that trigger new ideas or rekindle old ones. Without him, this calculus may well have been left unfinished, and this thesis would have been a boring read indeed.

A third person to thank is **Roger Antonsen**, who inspires by his fiery passion for the field of logic and mathematics. He is also responsible for the beautiful typography in this thesis.

A fourth person is **Sigurd Kvammen**, who has stood by me all the while during the writing of this thesis, listening patiently to me during my troubles despite being from a different field of science altogether. He also helped me proofread all the sixty+ pages of my work, catching typos and repetitive sentences with great agility. I also owe a thanks for proofreading to **Mads Thorstensen**.

Finally, my thanks go to the people at the logic seminar, who commented on early drafts of this work. Studying alongside you has been a pleasure and I hope to continue doing that.

Chapter 1

Introduction

This thesis is about automated theorem proving, a field of computer science that concerns itself with making machines able to reason in formal systems. An example of such a system is propositional logic. Another example is first-order logic, while a more familiar (to some) example is high school algebra. To accomplish this, computer scientists create calculi, sets of rules that determine what can be inferred from what is already there.

In this thesis we shall develop a calculus for the formal system known as coherent logic (CL), also called geometric logic. This system is a syntactic subset of first-order logic. There are already many types of calculi available for first-order logic; we adapt one of them, called Next-Generation Hyper-Tableaux, to coherent logic in the hopes of efficiency gains.

Coherent logic is usually defined as a fragment of first-order logic, where all formulae have the form

$$\forall \vec{x}.(A_1 \wedge \cdots \wedge A_n \rightarrow \exists \vec{u_1}.B_1 \vee \cdots \vee \exists \vec{u_m}.B_m)$$

where the A_i are atoms, while the B_i are conjunctions of atoms. (The notation $\vec{x}$ refers to a finite list of variables.) Additionally, there are no function symbols. Variables and constants are the only terms.

There has recently been a certain amount of interest in coherent logic [BC05, Bez05, dNM06], mostly for the following reasons:

- The natural axiomatization of many problems, for instance in axiomatic geometry, naturally lies in CL.
- For other problems, there is a satisfiability preserving translation into CL [Nor07] that does not need to introduce Skolem functions. As explained by Bezem and Coquand [BC05], this is a desirable property if proofs are to be translated into proof objects for a system like Coq.

- The very simple term structure makes it possible to use very efficient data structures, unification, etc.[1]

Proof procedures for CL typically use some variant of the hyper-tableau calculus, which requires all negative literals of a clause to be present before extending a branch. The question is what to do with universal variables in positive literals that might be split over branches. Normally, these variables are treated as universally quantified on the clause level, an approach that either leads to backtracking or, avoiding that, high memory consumption to keep the information that backtracking would discard. An example of this is found in the Hyper-Tableaux calculus (Section 2.1): If a literal Pxy is on the branch and we wish to expand with a clause $Pxx \rightarrow Qx$, the instantiation of x must be equal to that of y. One way to solve this is to expand and apply a global substitution $\{y/x\}$ on the tableau, but this may force you to backtrack if no refutation is reached. Alternatively, you apply a constraint to generate an expansion with $Qx << y = x$. The constraint means that the literal Qx may only be used in future expansions if you close your derivation by some substitution that unifies x and y. This avoids backtracking, but the constraints must be accumulated and propagated, a process that consumes memory.

If one wishes to avoid this, one can systematically instantiate such variables, commonly known as *critical variables*, with all ground terms, i.e. all existing constants. This is done in previous work on CL and this approach works, but it suffers from the need to guess the correct instantiation.

In this thesis we explore the possibility to treat these variables with an instance-based method, similar to the next-generation hyper tableau calculus of Baumgartner [Bau98]. Instance-based methods have proven to be effective in practice, e.g. in the work on disconnection by Letz and Stenz [LS07]. We are hoping for a further efficiency gain from the simple term structure of CL.

Previous work on instance-based methods is based on problems in clause normal form. To our knowledge the calculus presented in this thesis is the first instance-based calculus to work with a richer input.

1.1 Chapter guide

In Chapter 2 we give a comprehensive overview of related calculi, both instance-based and otherwise. We discuss their differences and similari-

[1] We should point out that the results of this thesis hold also if complex terms are permitted. However, there is then less motivation in considering the fragment.

ties, comparing them to our own calculus as well as to each other.

The presentation of our results starts with Chapter 3, which contains basic definitions used throughout this thesis. The calculus we have developed is presented and extensively discussed in Chapter 4, then proven sound and complete in chapters 5 and 6 respectively. Chapter 7 contains interesting things that we discovered but that didn't fit anywhere else. We conclude this thesis with a short summary and outline possible directions for future work in Chapter 8.

1.2 Scientific contributions

This work builds on both the CL calculus by Bezem and Coquand (Section 2.2) and the NG calculus by Baumgartner (Section 2.3.4). The contribution of this thesis consists of an instance-based hyper-tableau calculus for coherent logic that

- works on a richer input than clause normal form,
- does not introduce skolem functions,
- does not need any extra bookkeeping to handle existentially quantified variables, and
- stays sound and complete if complex terms are permitted. Also,
- finding rule applications can be done as efficiently as in the NG calculus despite the differences, and
- on problems in clause normal form, the calculus should perform as well as the NG calculus.

Chapter 2

Related calculi

In this chapter we shall look at different calculi related to the one presented in this work. As the calculus we have developed is an instance-based hyper-tableau calculus for coherent logic, it is related both to the original Hyper Tableaux described in Section 2.1 and the ground CL calculus described in Section 2.2. After these two calculi, we take a look at the field of instance-based methods in Section 2.3. Of the methods described, our calculus is an adaptation of the Next-Generation Hyper-Tableaux (NG) calculus described in Section 2.3.4.

2.1 Hyper Tableaux

Hyper tableau calculi work on formulae in clause normal form, and every clause is divided into a positive and a negative part. As an example, the clause $\neg Pxy \vee \neg R \vee Qxy$ becomes $Pxy \wedge R \rightarrow Qxy$ using this notation. In general, a clause is an implication $A \rightarrow B$ with A the *negative* and B the *positive* part.

In non-hyper tableau calculi, expanding a branch with a clause generates one new branch for every literal of the clause. Some of these new branches become closed, and some don't. In hyper-tableaux the general idea is different: An expansion with a certain clause C is not possible until every new branch containing a negative literal from C can be closed. Usually, this translates to 'the negative literals of C are already on the branch' in the ground case (cf. Section 2.2), or 'there exists a multiset unifier for the negative literals of C and the literals on the branch' in the general case. This idea is quite old, originally inspired by hyper-resolution. We quote from [Häh01, Sect. 4.6]:

> In fact, hyper tableaux were considered early on by Brown

[Bro78], but this work did not make the impact it deserved. The family of calculi known as model generation [MB88,FH91] is essentially a variant of hyper tableaux and will be discussed here as well.

In this section we shall look at a hyper-style tableau calculus of Baumgartner, Furbach, and Niemelä called Hyper Tableaux (HT) [BFN96]. This calculus is *not* instance-based itself, but acts as a foundation for the other instance-based calculi by Baumgartner.

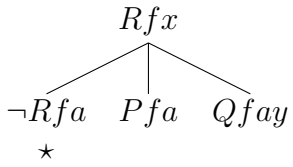
A key problem of HT is the problem of *critical variables*, i.e. variables that are split over positive literals of a clause, e.g. x in $S \rightarrow Pxa \vee Qxa$. A clause without such variables is called *pure*, and given a clause C and substitution σ, if $C\sigma$ is pure then σ is called a *purifying* substitution for C. We shall need these notions later.

The calculus constructs a tableau from a clause set by means of a single rule. Before describing this rule, we shall need the notion of a model for a tableau branch. In HT, all open branches contain only positive literals, and to any branch b the authors of [BFN96] associate a model $[b]$, namely the minimal Herbrand model that satisfies the universal closure of every literal on b. That this model satisfies a clause C is denoted $[b] \models C$.

We are now ready to describe the single rule of HT. Given a branch b and a clause $C = A \rightarrow B = \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$, in order to expand the tableau with C we need to find a substitution σ such that $[b] \models \forall(A\sigma)$, where $\forall(A)$ is the universal closure of A. This is called the *hyper condition*, and [BFN96] contains a method for finding such a substitution by syntactical means, more precisely by a terminating resolution procedure. The hyper condition essentially translates to 'after unification, all the negative literals of A are on the branch b', as is common for such calculi.

If such a substitution is found, the branch b is extended by $C\sigma\pi$ with π a purifying substitution for C, forming $n+m$ new branches. For every $i \leq n$ each new branch labeled with $\neg A_i\sigma\pi$ is closed, while for every $j \leq m$ each new branch labeled with $B_j\sigma\pi$ is open. The purifying substitution π allows us to treat variables in branch literals as universally quantified per literal, since no variable occurs in more than one literal.

Consider an example from [BFN96]: If we have a one branch tableau with literal Rfx on the branch plus a clause $Rx \rightarrow Px \vee Qxy$, the model for our branch satisfies $Rx\sigma$ with $\sigma = \{x/fx\}$. We can take a purifying substitution $\pi = \{x/a\}$ and get the tableau below (the closed branch is marked by a $\star$).



A major source of inefficiency in HT is the purifying substitution, and it is obsoleted in the NG calculus (see Section 2.3.4 for details). It is a source of inefficiency because it must be guessed more or less blindly, and we may in the worst case have to expand the tableau with every ground instance of a clause (try every possible purifying substitution) to achieve a refutation.

The authors of [BFN96] also present improvements over the basic calculus we just described, as well as experimental results on the prover they developed using HT.

2.2 CL calculus

There exists a calculus for coherent logic that we will call the CL calculus, developed by Bezem and Coquand [BC05] and described in more detail by Bezem in [Bez05]. The CL calculus is a hyper-tableau calculus, and requires all the negative literals of a clause to be present on a branch before using the clause to extend it. Additionally, the calculus is ground and works by enumerating every ground instance of every clause. Consider the example below, modified from [Bez05].

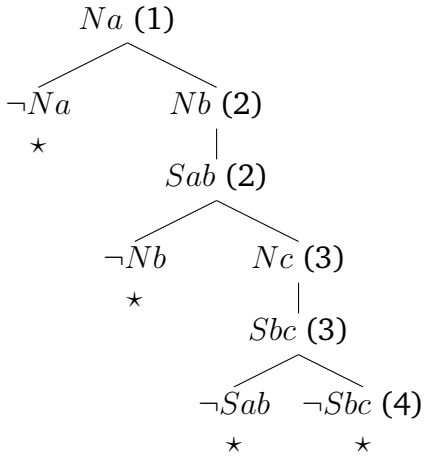
Clause set:

(A) Na

(B) $Nx \rightarrow \exists y(Ny \wedge Sxy)$

(C) $Sxy \wedge Syz \rightarrow \bot$

Derivation:



At the start, we have one ground term a, and we can expand with Na (1). We can also make a ground instance of (B) using a, and as Na is on the branch, we can expand with this instance. The existentially quantified variables are replaced with fresh constant symbols, yielding (2). Now we have a new constant b, and we can use it to make a new instance of (B). Expansion with this instance gives (3). Now we have enough symbols to make a ground instance of (C), namely $Sab \wedge Sbc$, and expand with it (4) to close the derivation.

The obvious deficiency of this method is that we can make many more instances that are not shown, and for every such instance we have to check if it can be used in an expansion step. To rectify this inefficiency, this work lifts the CL calculus to proper first-order coherent clauses. We do this by using unification to avoid enumerating ground terms, but *without* introducing skolem functions.

An interesting point worth mentioning is that while the satisfiability or validity of any coherent formula is decidable, satisfiability of coherent theories, i.e. sets of coherent formulae, is undecidable. This is proven in [Bez05] by reduction of the halting problem for register machines, known to be undecidable.

2.3 Instance-based methods

The general idea of instance-based theorem proving is to prove the unsatisfiability of a set of formulae in clausal normal form by generating sets of instances of these formulae, then checking the unsatisfiability of these sets. As any instance of a formula ϕ is a logical consequence of ϕ, if a set of instances is not satisfiable, neither is the original set.

Consider an example that does this in a very inefficient way. We generate instances of every clause in the clause set below by looping over possible constants to try, then check to see if the generated instances are unsatisfiable.

(A) Pxy

(B) $\neg Paz \vee Qaz$

(C) $\neg Pbz \vee Rbz$

(D) $\neg Rbc$

We could start by generating the set of instances

$$\{Paa, \neg Pab \vee Qab, \neg Pba \vee Rba, \neg Rbc\}$$

This set is satisfiable (make every positive literal true), so we continue the loop. Next, we can make the set

$$\{Pab, \neg Pab \vee Qab, \neg Pbc \vee Rbc, \neg Rbc\}$$

This set is also satisfiable, so we continue to find

$$\{Pbc, \neg Pac \vee Qac, \neg Pbc \vee Rbc, \neg Rbc\}$$

which is unsatisfiable, and we can stop.

There are two important points of inefficiency here. The first one is that the clause $\neg Paz \vee Qaz$ is superfluous, as the clause set is unsatisfiable without it. We should thus avoid generating instances of this clause altogether. The second point of inefficiency is the lack of direction in the instantiation process, leading to many dead-ends as we generate instances that do not lead to an unsatisfiable set. Usually, the sets are not discarded and thus there are no dead-ends, but one still has to deal with all the superfluous instances generated.

Several improvements on the basic idea that we presented above have been made. A general idea is to use unification of complementary literals to guide the instantiation process, done in a very basic form in the Inst-Gen calculus of Ganzinger and Korovin [Kor]. In the Hyper-Linking calculus of Lee and Plaisted [LP92], the instantiation process, and to an extent the choice of clauses to instantiate, is guided by the simultaneous unification of all complementary literals. This continues in the Next-Generation Hyper Tableau calculus of Baumgartner et al. [Bau98], which we use as a basis for our own calculus. There, a tableau derivation is used to much more actively guide the choice of clause to instantiate next while still using unification to find the instance needed. A similar idea motivates the Disconnection calculus of Billon [Bil96], improved upon by Letz and Stenz [LS07].

A different approach, focusing less on the syntax of the clauses and more on the semantics of a hypothetical model for them is adopted in the FDPLL [Bau00] and Model Evolution [BT03] calculi, also by Baumgartner (the latter together with Tinelli). We will discuss all of these in order, then conclude this section with a short summary.

A common feature of instance-based methods is their ability to decide the Bernays-Schönfinkel class of problems, also known as 'essentially propositional' problems. A problem in this class is a set of formulae of the form $\exists\vec{x}\forall\vec{y}.\phi(\vec{x},\vec{y})$, where $\phi(\vec{x},\vec{y})$ contains no quantifiers or function symbols. To solve such a problem is to decide whether or not the set is satisfiable, and the fact that instance-based methods *decide* this class

of problems means that they terminate on any such problem with a correct answer. This class of problems is represented at CASC, the CADE ATP Systems competition [SS06], which is a major competition for automated theorem provers. The winners in this class (called 'EPR') are usually instance-based provers.

Before proceeding to the individual methods we shall require a definition used by several of them. Note that some of the calculi below do not refer to links directly, but use the concept nonetheless.

Definition 2.1 (Link) *Given two distinct clauses C and D, a link between C and D is a pair of literals $L \in C$ and $\neg K \in D$ such that there exists a unifier for L and K.* □

2.3.1 Inst-Gen

This calculus, developed by Korovin and Ganzinger, can be seen as the 'smallest' instance-based method. Several of the calculi mentioned later precede the papers we refer to here, but if one ignores the timeline, Inst-Gen is the 'first' method in the sense that nearly everything else can be seen as an extension of it. It is presented in [GK03], but our reference for this section is [Kor].

The calculus works by looking for links within clauses. If a link between two clauses $C \vee L$ and $D \vee \neg K$ with unifier σ is found, and σ instantiates L or $\neg K$, then clauses $(C \vee L)\sigma$ and $(D \vee \neg K)\sigma$ are added to the current clause set. One then replaces every variable in every clause in the clause set by a special constant and checks the result for satisfiability, much as in the unguided calculus we described at the beginning of Section 2.3. The beauty of Inst-Gen is that it is 'trivially' sound and easy to prove complete — sound because we always add instances of clauses already present, hence consequences thereof, and complete by a simple proof using Herbrand's theorem. Also, it is extensible by many refinements, which are discussed in [Kor]. These include hyper-style inference, semantic selection (add instances that are not yet satisfied by a generated model for the clauses we do have), and redundancy elimination.

In addition to the work mentioned above, [GK04] presents an integration of equality reasoning into Inst-Gen, while [Kor08] is a system description of iProver, a theorem prover based on an improved version of Inst-Gen.

2.3.2 Hyper-Linking

Developed by Lee and Plaisted around 1990 (we use [LP92] as our main reference for this section), this is one of the first instance-based methods. The Hyper-Linking calculus works, similarly to Inst-Gen, by looking for links between the clauses. The process is divided into 'rounds'; each round computes for every clause C the set of hyper-links for this clause.

Definition 2.2 (Hyper-link) *Given a clause $C = \{L_1, \ldots, L_n\}$, a hyper-link for C is a set of links such that for every i, there is one and only one link featuring L_i.* □

For every such hyper-link, we can compose the unifiers of every link in it (possibly renaming variables) to get a single substitution for the hyper-link. Call this substitution θ; the instance we add to our clause set is $C\theta$. After all the clauses have been processed (this process terminates), we can ground every clause by substituting some distinct constant for every variable to obtain a set of ground clauses. This set can be checked for satisfiability by e.g. resolution, and if it is refuted we are done. Otherwise, a new round is initiated with the clause set we failed to refute as input.

In this calculus, the instances computed attempt to create clauses that are complementary. This helps to avoid superfluous clauses, as they would have few or no links to other clauses. However, if there is a 'cluster' of superfluous clauses, i.e. clauses that have links between them but no links to other clauses, and this cluster is satisfiable, Hyper-Linking would still generate instances of clauses in the cluster.

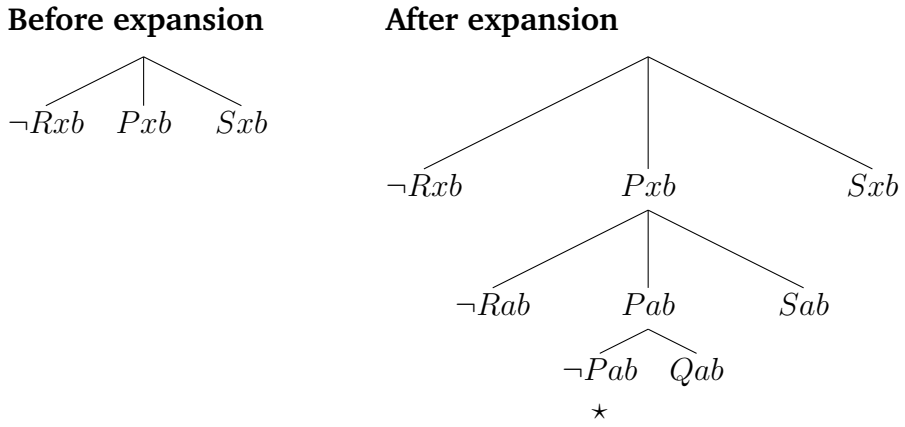
A lot of work on Hyper-Linking has been done since its inception. In 1990, a prover called CLIN was developed by Lee [Lee90] for this calculus. Around 1992, Alexander and Plaisted integrated equality reasoning into the calculus [AP92, Ale95], and Alexander later developed a prover CLIN-E [Ale97] to address inefficiencies in CLIN that he and Plaisted had uncovered during their equality research.

A parallel line of research looked for improvements to Hyper-Linking itself, by adding a candidate model to the proof search and using Hyper-Linking to find instances of clauses that are false in this model. If such an instance is found, the model is modified to satisfy it. The first attempt [CP94], due to Chu and Plaisted, came around 1994, and a prover for the method [CP97] was developed. This was followed by [PZ00] and finally by [YP02], which left the Hyper-Linking paradigm.

2.3.3 Disconnection

The Disconnection calculus was conceived by Billon [Bil96] and improved upon by Letz and Stenz [LS07], whose paper is our main reference for this section. They also developed a prover for this calculus called DCTP [LS01], and integrated equality reasoning into the calculus (covered in [LS07], but originally in [LS02]).

One big difference to Hyper-Linking is that Disconnection is a tableau calculus. However, unlike the calculi we shall be discussing later, this calculus also works on links (sometimes called connections). In contrast to Hyper-Linking, where all the links of a clause are used together, Disconnection looks at one link at a time. The general idea is as follows: We draw an initial path through the current clause set to act as a fake branch at the start of the derivation. Then, if we have two clauses C and D on a branch or in the current clause set, and have found a link between C and D using literals on the (real or fake) branch, we expand the tableau by clauses $C\sigma$ and $D\sigma$ using the unifier σ for this link. Consider an example expansion with the clause $\neg Paz \vee Qaz$ with $\neg Paz$ being the literal on the initial path:



To use the link between Pxb and $\neg Paz$ we have to instantiate x to a and z to b. Before doing the expansion with $\neg Paz \vee Qaz$, we put the instantiated tableau clause on the branch. The branch marked by $\star$ is closed, as it contains complementary literals (Pab and $\neg Pab$) if grounded to the constant $*$ (vacuous in this case since the literals are ground). Any (real) branch satisfying that condition can be closed, and there is always one such branch in every expansion.

This way of looking for single links that are 'needed', i.e. can be used together with the clauses we already have, removes the problem of superfluous clauses to a large extent. If you are unlucky, you can still end up de-

riving a cluster of superfluous clauses, but not necessarily. Hyper-Linking, on the other hand, does not have a way to avoid this possibility. On the negative side, Disconnection generates many similar branches, something that can lead to near-copies of derivations.

2.3.4 Next-Generation Hyper Tableaux

The NG calculus of Baumgartner [Bau98] is an instance-based version of his Hyper Tableaux calculus [BFN96], developed together with Furbach and Niemelä. The Hyper Tableaux calculus was partly ground: If a variable occurred in two different literals in the positive part of a clause, like x in $Pa \rightarrow Qxy \vee Rx$, HT would guess a ground term to substitute for x (see Section 2.1). The NG calculus improved this by adding proper unification support and thus removing this inefficiency.

The NG calculus is also a tableau calculus, but is only indirectly concerned with links. Instead, we divide all clauses into a positive and a negative part, so that the clause $\neg Pax \vee \neg Qax \vee Rbx$ becomes $Pax \wedge Qax \rightarrow Rbx$. The calculus works by looking for the situation where the negative side of a clause unifies with the literals on a branch — in other words, a situation where every negative literal of a clause is part of a link with some literal on the branch. Then, two things can happen: If the composed substitution of this set of links is such that no branch literal is instantiated, we can expand the branch using our clause. When this happens, every new branch containing a literal from the negative side of a clause is closed. If the condition above is not true, we do not expand the branch using our clause. Instead, we create instances of every clause on the tableau that participates in the set of links using the substitution we computed, then add all these instances to our clause set.

This calculus gets much the same benefits as Disconnection, and generates a similar amount of branches, albeit in a slightly different order. This is expected, as a branch expansion in NG can be thought of as a list of Disconnection expansions, one for every negative literal found in the clause (modulo order). The generation of instances that NG does explicitly is done implicitly in Disconnection, cf. Section 2.3.3. The two calculi are also very similar in their definitions of redundant inferences and model generation from an open branch.

The calculus presented in this thesis is an instance-based version of the CL calculus (see Section 2.2) based on the NG calculus. The CL calculus can be seen as a ground version of NG, and our work is thus a lifting of that to free-variable clausal form. On the other hand, this work can be seen as an extension of NG that enables it to handle clauses with two

types of variables. For examples, as well as a discussion of the differences between NG and our own calculus, we refer the reader to Chapter 4.

Finally, there is a paper by Feng, Sun, and Wu [FSW06] that claims to have found a counterexample to the completeness of NG. This counterexample is erroneous; we discuss it in Section 6.1. As such, we believe that NG is indeed complete as proven in [Bau98].

2.3.5 FDPLL

Despite the fact that NG is complete, provers based on it have not performed as expected. This fact prompted Baumgartner to try a more semantic approach, and the FDPLL calculus [Bau00] is the first result from this line of research. It attempts to lift the propositional DPLL procedure to a first-order calculus, as DPLL has performed quite well on propositional problems.

DPLL works by case analysis: Given a clause set S, we pick a propositional variable, say A, from a clause in S, and create two new clause sets $S[A/\top]$ and $S[A/\bot]$ (by $S[A/\bot]$ we mean the set S with every instance of A replaced by $\bot$), to be analyzed separately. The two clause sets we created are simpler than S, and can be further simplified by propositional rules, e.g. $A \vee \top \equiv \top$. If we find an elementary contradiction during this simplification, that clause set is unsatisfiable. If not, we pick a new variable to split on until we either run out of propositional variables (in case which we conclude that S is satisfiable), or find that all the sets we have generated are unsatisfiable, which means that S is not satisfiable either.

FDPLL lifts this splitting rule to non-ground literals by trying to generate a potential model for a set S of non-ground clauses. This is a tableau calculus, so the usual notions of branches etc. apply. At the beginning, the potential model satisfies nothing, i.e. assigns false to every instance of every positive literal, and the single branch has a 'literal' $\neg x$ on it that unifies with any positive literal. We then look for a clause $C \in S$ which is falsified by the model, a statement equivalent to 'every literal of C unifies with the complement of a branch literal using unifier σ'. If it is then true that $C\sigma$ contains a literal L such that neither L nor the complement of L is on the branch, we split on this literal and create two new branches, equivalent to two potential models: One satisfying every instance of L and another one falsifying every such instance. In general, a literal on the branch represents every instance of itself, except when there is a more specific literal on the branch generating an exception. This is similar to our own model construction, cf. Def. 6.8.

There is a second rule in FDPLL, the Commit rule. After a split, a

branch may contain literals L and $\neg K$ that unify with a unifier σ, such that neither $L\sigma$ nor the complement is on the branch. This is unfortunate for the model generation, as we have no way to decide what to satisfy. The Commit rule remedies the situation by creating two new branches: One with $L\sigma$ and another with the complement.

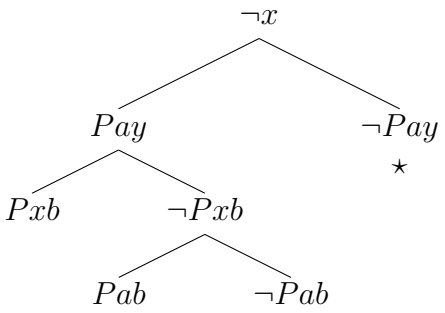
Finally, a branch is closed if there exists a clause D and substitution δ such that for every $L \in D$, the complement of $L\delta$ is on the branch. To make more sense of this description, consider a simple example taken from [Bau00].

Clause set:

(A) Pay

(B) $Pxb \vee \neg Pzy \vee Qxyz$

Derivation:



We start with the catch-all $\neg x$, then unify clause A with $\neg x$ using the empty substitution to split on Pay. The right branch is closed due to clause A. Next, we can unify B with the open branch by unifying $\neg Pzy$ with Pay and the rest with $\neg x$. Neither the literal Pxb nor the complement is on the branch, so we can split again. Finally, we can see the Commit rule being applied: As Pay and $\neg Pxb$ unify to Pab, we can use Commit to get the two branches with Pab and $\neg Pab$.

2.3.6 Model Evolution

A continuation of Baumgartner's work on FDPLL, the Model Evolution calculus (MEV) is 'a more faithful lifting of the DPLL procedure' [BT03, abstract]. Here, the focus is also on the model, instead of the clauses themselves. A sequent calculus, MEV operates on sequents of the form $\Lambda \vdash \Phi$. Λ is called a context, while Φ is the clause set we are trying to refute. The procedure is started on a clause set S by constructing the sequent $\neg v \vdash S$, with $\neg v$ a catch-all metaliteral similar to the one used in FDPLL. If we manage to derive a sequent $\Lambda \vdash \emptyset$ from $\neg v \vdash S$, then S is satisfiable, and the context Λ gives a model for S (in general we will never reach this sequent, as the derivation becomes infinite). On the other hand, if we manage to derive a sequent $\Lambda \vdash \Phi, \square$ where $\square$ is the empty

clause, the branch containing this sequent is closed and cannot lead to a model for S.

The chief advantage of MEV is that it manages to lift the simplification rules of DPLL to first-order logic. An example of this is the Subsume rule, which in the propositional case allows one to simplify $\{L, L \vee C, \ldots\}$ to $\{L, \ldots\}$ — as L has to be true, the satisfiability of this clause set does not depend on $L \vee C$. We say that L *subsumes* $L \vee C$. Such rules can save a lot of work, and MEV contains two of them, the Subsume rule mentioned above, and a second rule called Resolve.

In addition to the simplification rules there are two rules that add literals to the context, Assert and Split. The Split rule is similar to the splitting rule in FDPLL, while the Assert rule is a lifting of the propositional rule that assigns $\top$ to a clause containing a single literal, called a *unit clause*. Such a clause can only be made true in one way, namely by assigning true to the literal (in the propositional case), or assigning true to every instance of the literal (in the first-order case).

To achieve a sound and complete lifting of these rules, the authors of [BT03] introduce the notion of a *parameter*, a type of variable they use to constrain the context. The general idea here as well is that a literal L in context Λ represents every instance of itself, unless a more specific negative literal creates an exception. Here, parameters come into play: If L is parameter-free, it will not be subject to exceptions, potentially leading to an efficiency gain. New parameters are introduced and propagated via the Split rule, and care is taken in all rules to avoid contradictions in the context.

Quite a lot of work has been done on MEV. A theorem prover called DARWIN, two-times winner of the 'EPR' problem class of CASC [SS06], has been developed by Fuchs, together with Baumgartner and Tinelli [BFT04, Fuc04]. The calculus has also been improved by 'lemma learning' [BFT06], a method for generating formulae to avoid dead-ends later in the search. A further and more general improvement was found by Fuchs, Baumgartner, Tinelli, and de Nivelle [BFdNT09].

2.3.7 Summary

The calculi presented here give a short overview of the current knowledge of instance-based methods. For a more in-depth tutorial, we refer the reader to the IJCAR 2004 tutorial on instance-based methods [BS].

Chapter 3

Preliminaries

In this chapter we shall define coherent logic as a subset of first-order logic in Section 3.1. Then, in Section 3.2 we shall define a clausal coherent logic similar to clause normal form, but with two types of variables. We give a semantics for it in Section 3.2.1, then we prove Theorem 3.16 stating that the clausal logic is an accurate representation of coherent logic. The chapter is concluded by Section 3.2.2, which contains definitions of various substitutions and relations to be used throughout this thesis.

3.1 First-order logic

Definition 3.1 (Signature) *A* signature $\Sigma = \langle \mathcal{P}, \mathcal{V}, \mathcal{C} \rangle$ *is a tuple with* $\mathcal{P}$ *an infinite set of* predicate symbols*,* $\mathcal{V}$ *an infinite set of* variables*, and* $\mathcal{C}$ *an infinite set of* function symbols*.*

A signature is coherent *if the function symbols are all of arity zero, i.e. constants.*

The set of Σ-terms $\mathcal{T}$ *is the union* $\mathcal{V} \cup \mathcal{C}$*, hence a term is either a variable or a constant.* □

Definition 3.2 (Formulae) *Given a coherent signature* $\Sigma = \langle \mathcal{P}, \mathcal{V}, \mathcal{C} \rangle$*, the set of* Σ*-formulae* $\mathcal{F}$ *is the least set such that*

1. $P(t_1, \ldots, t_n) \in \mathcal{F}$ *for* $P \in \mathcal{P}$ *and* $t_1, \ldots, t_n \in \mathcal{T}$
2. $\neg\phi \in \mathcal{F}$ *for* $\phi \in \mathcal{F}$
3. $\phi \circ \psi \in \mathcal{F}$ *for* $\circ \in \{\wedge, \vee, \rightarrow\}$ *and* $\phi, \psi \in \mathcal{F}$
4. $\forall x.\phi$ *and* $\exists x.\phi \in \mathcal{F}$ *for* $x \in \mathcal{V}$ *and* $\phi \in \mathcal{F}$

Any formula $P(t_1, \ldots, t_n)$ above is called atomic *or simply an* atom. *We will usually omit the parentheses and write Pxy for $P(x, y)$ wherever it is convenient.* □

We treat $\wedge$ and $\vee$ as commutative and associative, thus writing $A_1 \wedge \cdots \wedge A_n$ without parentheses. The connective $\rightarrow$ has the lowest precedence and $\neg$ the highest. Also, we assume that quantifiers bind free variables in formulae, thus in the formula $\forall x.\phi$, x occurs free in ϕ. Given this, we can define what this thesis is about, namely coherent formulae.

Definition 3.3 (Coherent formula) *A* coherent formula *is any formula of the form*

$$\forall \vec{x}.(A_1 \wedge \cdots \wedge A_n \rightarrow \exists \vec{u_1}.B_1 \vee \cdots \vee \exists \vec{u_m}.B_m)$$

with atomic formulae A_i and conjunctions of atomic formulae B_j for $i \leq n$ and $j \leq m$. (The notation $\vec{x}$ refers to a finite list of variables.) We allow the cases $n = 0$ and $m = 0$, writing $\top \rightarrow \phi$ (or just ϕ) and $\phi \rightarrow \bot$ respectively. □

A coherent formula is thus a universally quantified implication from a conjunction of atoms to a disjunction of existentially quantified conjunctions of atoms.

3.1.1 First-order semantics

A model $\mathcal{M}$ for a logical language consists of a set $|\mathcal{M}|$, called the domain, and an interpretation function $s^{\mathcal{M}}$ for s a predicate or constant symbol. We demand that every constant be interpreted as an element of $|\mathcal{M}|$ and every predicate symbol of arity n as a relation on $|\mathcal{M}|^n$. To interpret free variables, we shall use an *assignment*.

Definition 3.4 (Assignment) *An assignment for a model $\mathcal{M}$ is a function from variables to elements of $|\mathcal{M}|$. We use μ, η, and θ to denote assignments.* □

Example 3.5 *Let $\mathcal{M}$ be a model with domain $|\mathcal{M}| = \{a, b, c\}$. The assignment $\mu = \{x/a, y/a, z/c\}$ maps x and y to a and z to c, so we have that $\mu(x) = a$ and $\mu(z) = c$, while the value of $\mu(w)$ is undefined.* □

Free variables are thus interpreted as domain elements by the assignment. Given this, a formula ϕ is true in $\mathcal{M}$ under μ, equivalently $\mathcal{M}$ satisfies ϕ under μ, which we write as $\mathcal{M}, \mu \models \phi$, as defined below (modified from [Han04, pp. 9–10]):

1. For atomic formulae, $\mathcal{M}, \mu \models P(a_1, \ldots, a_n)$ if $\langle a_1^{\mathcal{M},\mu}, \ldots, a_n^{\mathcal{M},\mu} \rangle \in P^{\mathcal{M}}$.
2. $\mathcal{M}, \mu \not\models \bot$.
3. $\mathcal{M}, \mu \models \neg\phi$ if $\mathcal{M}, \mu \not\models \phi$.
4. $\mathcal{M}, \mu \models \phi \wedge \psi$ if $\mathcal{M}, \mu \models \phi$ and $\mathcal{M}, \mu \models \psi$.
5. $\mathcal{M}, \mu \models \phi \vee \psi$ if $\mathcal{M}, \mu \models \phi$ or $\mathcal{M}, \mu \models \psi$.
6. $\mathcal{M}, \mu \models \phi \rightarrow \psi$ if $\mathcal{M}, \mu \models \phi$ does not hold or $\mathcal{M}, \mu \models \psi$ holds.
7. $\mathcal{M}, \mu \models \exists x.\phi$ if there exists an assignment μ' differing from μ only in the value assigned to x, such that $\mathcal{M}, \mu' \models \phi$.
8. $\mathcal{M}, \mu \models \forall x.\phi$ if for every possible assignment μ', differing from μ only in the value assigned to x, $\mathcal{M}, \mu' \models \phi$ holds.

A formula ϕ without free variables is true in a model $\mathcal{M}$ if it is true in that model under an arbitrary assignment (as the assignment does nothing in this case), which we write simply as $\mathcal{M} \models \phi$.

3.2 Clause logic

The calculus presented in Chapter 4 does not work on formulae, but on clauses. Syntax and semantics for these are defined in this section. First of all, we are going to change our definition of variables and terms, as we distinguish between universal and existential variables instead of having explicit quantifiers. We also distinguish between the constants that come with the problem and witnesses introduced by instantiating existential variables. There are still no (non-nullary) functions, so the only terms are variables, constants, and witnesses. The definitions in this chapter will be used throughout the rest of this thesis — thus wherever we speak of variables, terms, etc., we use the definitions from this section.

Definition 3.6 (Variables, terms) *Given a coherent signature from Def. 3.1, we divide the set of all* variables $\mathcal{V}$ *into two disjoint sets: the set of universal variables $\mathcal{V}^a$ and the set of existential variables $\mathcal{V}^e$. We will use x, y, z to denote universal variables, and a dot to mark existential ones, e.g. $\dot{x}$.*

The type *of a variable is either existential or universal. The type of a variable x is universal iff $x \in \mathcal{V}^a$, otherwise the type of x is existential.*

We keep the set $\mathcal{C}$ of constant symbols *as before, but add an infinite set $\mathcal{W}$ of* witness symbols *which are used like constants, but are not in $\mathcal{C}$. We will use a, b, c for constants, and denote witnesses by a bar, e.g. $\bar{u}$.*

The set of all terms *is defined as the union $\mathcal{T} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{W}$.* □

Example 3.7 *If P is a predicate symbol, then $Px\dot{x}$ is a literal with universal variable x and existential variable $\dot{x}$. Likewise, Pax is a literal with a constant a and universal variable x, while $P\bar{u}x$ is a literal with a witness $\bar{u}$ and universal variable x.*

As there are no function symbols, expressions like $Pf(x)\dot{x}$ and $Paf(\dot{u})$ are not literals. □

Next, we can define clauses and clause sets, as well as the relation between clauses and formulae.

Definition 3.8 (Clauses, clause sets) *A* coherent clause *or simply* clause *is a multiset of literals and sets of literals, written as a disjunction $\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n$ where $m, n \geq 0$. We may also write this as $A \to B$, where $A = \{A_1, \ldots, A_m\}$ is called the* negative part *or the* negative literals *of the clause, and $B = \{B_1, \ldots, B_n\}$ the* positive part*. The elements of A are atoms with no existential variables, while the elements of B are* sets *of atoms that, while they may contain both types of variables, must be pairwise disjoint in their existential variables. Clauses with B non-empty are called* program clauses*.*

The literals *of $A \to B$, written $Lits(A \to B)$, are the negated atoms of A together with all the atoms in every element of B, i.e the union $\{\neg L | L \in A\} \cup \bigcup B$.*

The elements *of $A \to B$, written $Elems(A \to B)$, are the negated atoms of A and the elements of B, i.e. the union $\{\neg L | L \in A\} \cup B$.*

A clause set *is a set of clauses.* □

Definition 3.9 (Formulae and clauses) *To any coherent formula ϕ we associate a coherent clause ϕ^c as follows:*

1. *If ϕ is atomic, then $\phi^c = \phi$.*
2. *If ϕ is a conjunction of atoms $A_1 \wedge \cdots \wedge A_n$, then $\phi^c = \{A_1^c, \ldots, A_n^c\}$.*
3. *If ϕ is a disjunction of atomic conjunctions $\exists \vec{x}_1 . B_1 \vee \cdots \vee \exists \vec{x}_m . B_n$, then $\phi^c = \{{B_1}^c, \ldots, {B_n}^c\}$. Every variable bound by an existential quantifier is in $\mathcal{V}^e$.*

4. *If ϕ is an implication $\forall\vec{x}(A \to B)$, then $\phi^c = A^c \cup B^c$. Every variable bound by a universal quantifier is in $\mathcal{V}^a$.*

In other words, any coherent formula can be turned into a coherent clause by removing the quantifiers and transforming conjunctions and disjunctions into sets. □

Example 3.10 *If P, Q, R, and S are predicate symbols, then both*

$$Pxy \wedge Qab \to R\dot{u}c$$

and

$$(Qx\dot{u} \wedge Ry\dot{v}) \vee Sa\dot{w}$$

are clauses, corresponding to coherent formulae

$$\forall x \forall y(Pxy \wedge Qab \to \exists\dot{u}(R\dot{u}c)$$

and

$$\forall x \forall y \exists\dot{u} \exists\dot{v}((Qx\dot{u} \wedge Ry\dot{v}) \vee \exists\dot{w}(Sa\dot{w})$$

respectively. On the other hand, neither

$$P\dot{u}x \to Qab$$

nor

$$Pxy \to Qx\dot{u} \vee Ry\dot{u}$$

are clauses. The first expression contains an existential variable in the negative part, while the literals in the positive part of the second are not disjoint in their existential variables. □

Example 3.11 *Given the clause $C = Pxy \wedge Qxy \to R\dot{u}x \wedge S\dot{u}y$, we have that the literals of C are $Lits(C) = \{\neg Pxy, \neg Qxy, R\dot{u}x, S\dot{u}y\}$ and that the elements of C are $Elems(C) = \{\neg Pxy, \neg Qxy, \{R\dot{u}x, S\dot{u}y\}\}$.* □

3.2.1 Clausal typed-variable semantics

When we move from coherent formulae to coherent clauses all variables become free, and the ordinary first-order semantics no longer suffice. Typed-variable semantics introduce the notions of universal and existential assignments (since we have corresponding variables) to interpret free variables.

Definition 3.12 (Universal and existential assignment) *A* universal assignment *is a function from universal variables to domain elements, while an* existential assignment *is a function from existential variables to domain elements. We will use a superscript 'a' to show that an assignment is universal, and a superscript 'e' to show that it is existential.* □

Given a model $\mathcal{M}$, a universal assignment μ^a and an existential assignment μ^e, we define the interpretation of a term s in the model $\mathcal{M}$ under universal assignment μ^a and existential assignment μ^e, written $s^{\mathcal{M},\mu^a,\mu^e}$, as follows:

1. $c^{\mathcal{M},\mu^a,\mu^e} = c^{\mathcal{M}}$ for a constant symbol c.
2. $x^{\mathcal{M},\mu^a,\mu^e} = \mu^a(x)$ for a universal variable x.
3. $\dot{u}^{\mathcal{M},\mu^a,\mu^e} = \mu^e(\dot{u})$ for an existential variable $\dot{u}$.

Example 3.13 *Consider a model $\mathcal{M}$ where every constant is interpreted as itself with domain $|\mathcal{M}| = \{a, b, c\}$, together with universal assignment $\mu^a = \{x/a, y/a, z/b\}$ and existential assignment $\mu^e = \{\dot{u}/c, \dot{v}/c\}$. The literal $Px\dot{u}a$ is then interpreted as $(Px\dot{u}a)^{\mathcal{M},\mu^a,\mu^e} = P^{\mathcal{M}}\mu^a(x)\mu^e(\dot{u})a^{\mathcal{M}} = P^{\mathcal{M}}aca$.* □

Predicate symbols are interpreted as relations, the same as in the first-order semantics. We can now define $\mathcal{M}, \mu^a, \mu^e \models \phi$ (ϕ is true in $\mathcal{M}$ under μ^a and μ^e) for a clause ϕ as follows:

1. If ϕ is an atomic formula $P(t_1, \ldots, t_n)$, then $\mathcal{M}, \mu^a, \mu^e \models P(t_1, \ldots, t_n)$ if $\langle t_1^{\mathcal{M},\mu^a,\mu^e}, \ldots, t_n^{\mathcal{M},\mu^a,\mu^e} \rangle \in P^{\mathcal{M}}$.
2. If ϕ is a set of atoms, then $\mathcal{M}, \mu^a, \mu^e \models \phi$ if $\mathcal{M}, \mu^a, \mu^e \models L$ for every $L \in \phi$.
3. If ϕ is of the form $A \rightarrow B$, then $\mathcal{M}, \mu^a, \mu^e \models \phi$ if $\mathcal{M}, \mu^a, \mu^e \not\models A$ or $\mathcal{M}, \mu^a, \mu^e \models S$ for some $S \in B$.

A clause ϕ is true in a model $\mathcal{M}$ ($\mathcal{M}$ satisfies ϕ) if for every universal assignment μ^a there exists an existential assignment μ^e such that $\mathcal{M}, \mu^a, \mu^e \models \phi$. For a clause ϕ without variables, this collapses to $\mathcal{M} \models \phi$ in the ordinary first-order semantics. A clause is satisfiable if there exists a model that satisfies it.

A clause set is satisfiable if there exists a model that satisfies every clause in the set. If no such model exists, the clause set is unsatisfiable (contradictory).

Example 3.14 *Let $\mathcal{M}$ be a model that interprets every constant as itself, with $|\mathcal{M}| = \{a, b, c\}$. Let P and Q be predicate symbols, and define the interpretations $P^{\mathcal{M}} = \{\langle a, b\rangle, \langle b, c\rangle\}$ and $Q^{\mathcal{M}} = \{\langle c, b\rangle, \langle a, c\rangle\}$.*

Consider a clause $C = Pxy \wedge Qx\dot{u}$. This clause is a set of atoms, and it is true *in $\mathcal{M}$ under the assignments $\mu^a = \{x/a, y/b\}$ and $\mu^e = \{\dot{u}/c\}$, as $\langle a, b\rangle \in P^{\mathcal{M}}$ and $\langle a, c\rangle \in Q^{\mathcal{M}}$. However, under the assignments $\eta^a = \{x/c, y/a\}$ and $\eta^e = \{\dot{u}/b\}$ the clause C is* false*, as $\langle c, a\rangle \notin P^{\mathcal{M}}$, which means that $\mathcal{M}, \eta^a, \eta^e \not\models Pxy$.* □

Example 3.15 *The clause set*

$$\{Pxy, Pxy \to Rab\}$$

is satisfiable, as it can be satisfied by a model $\mathcal{M}$ such that $\mathcal{M} \models Pxy$ and $\mathcal{M} \models Rab$. On the other hand, the clause set

$$\{Pxy, Pxy \to Rab, Rab \to \bot\}$$

is not satisfiable, as any potential model would have to satisfy $\bot$, which is not possible. □

Apart from the split between universal and existential assignment, the semantics is quite standard. That the clausal semantics are correct is captured in the following theorem.

Theorem 3.16 (Equivalence of semantics) *For every model $\mathcal{M}$ and coherent formula ϕ with corresponding coherent clause ϕ^c, $\mathcal{M} \models \phi$ if and only if $\mathcal{M} \models \phi^c$.* □

Proof *For the if part, assume that $\mathcal{M} \models \phi^c$. This means that for every universal assignment μ^a there exists an existential assignment μ^e such that $\mathcal{M}, \mu^a, \mu^e \models \phi^c$. Let μ^a be an arbitrary universal assignment, and let μ^e be the corresponding existential assignment. We can compose them to form an assignment $\mu = \mu^a\mu^e$.*

Now we shall prove by induction that $\mathcal{M}, \mu \models \phi$. For the base case, if ϕ is atomic, $\mathcal{M}, \mu \models \phi$ since $\phi = \phi^c$.

For the general case, if ϕ is a conjunction of atoms $A_1 \wedge \cdots \wedge A_n$, then $\phi^c = \{A_1^c, \ldots, A_n^c\}$. By assumption, $\mathcal{M}, \mu^a, \mu^e \models A_i^c \in \phi^c$ for every i, and by induction $\mathcal{M}, \mu \models A_i$ for every i, as the A_i are atomic. This, however, means that $\mathcal{M}, \mu \models \phi$ by the ordinary FOL semantics.

If ϕ is an existentially quantified disjunction of conjunctions of atoms $\exists\vec{x}_1.B_1 \vee \cdots \vee \exists\vec{x}_n.B_n$, then $\phi^c = \{B_1^c, \ldots, B_n^c\}$. By assumption, $\mathcal{M}, \mu^a, \mu^e \models B_i^c \in \phi^c$ for some i, and by induction, $\mathcal{M}, \mu \models B_i$ for some i. This means

that there are elements in $|\mathcal{M}|$ that can be assigned to the variables $\vec{x}_i$ such that B_i is true in $\mathcal{M}$ under μ, and that again gives us that $\mathcal{M}, \mu \models \exists \vec{x}_i . B_i$. However, this means that $\mathcal{M}, \mu \models \phi$.

Finally, if ϕ is an implication $\forall \vec{x}(A \rightarrow B)$, then $\phi^c = A^c \cup B^c$. By assumption, either $\mathcal{M}, \mu^a, \mu^e \not\models A^c$ or $\mathcal{M}, \mu^a, \mu^e \models S$ for some $S \in B^c$. If $\mathcal{M}, \mu^a, \mu^e \not\models A^c$ holds, then by induction $\mathcal{M}, \mu \not\models A$, which means that $\mathcal{M}, \mu \models A \rightarrow B$. This means that there are elements in $|\mathcal{M}|$ that can be assigned to the variables $\vec{x}$ such that $A \rightarrow B$ is true in $\mathcal{M}$ under μ, which gives us that $\mathcal{M}, \mu \models \forall \vec{x}(A \rightarrow B)$.

If $\mathcal{M}, \mu^a, \mu^e \models S$ for some $S \in B^c$ was the case, then by induction $\mathcal{M}, \mu$ satisfies some disjunct of B, and the rest of the argument proceeds exactly as in the paragraph above.

Since μ^a was arbitrarily chosen, this holds for every such assignment, which means that $\mathcal{M} \models \phi$ for every assignment μ, which was the desired property.

For the only if part, we shall assume that $\mathcal{M} \models \phi$. This means that for every assignment μ, $\mathcal{M}, \mu \models \phi$. Let μ be arbitrarily chosen, and split it into a universal part μ^a and an existential part μ^e. The proof is also by induction on the form of ϕ. For the base case, if ϕ is atomic, $\mathcal{M}, \mu^a, \mu^e \models \phi^c$ since $\phi = \phi^c$.

If ϕ is a conjunction of atoms $A_1 \wedge \cdots \wedge A_n$, then $\phi^c = \{A_1^c, \ldots, A_n^c\}$. By assumption, $\mathcal{M}, \mu \models A_i$ for every i, and by induction, $\mathcal{M}, \mu^a, \mu^e \models A_i^c$ for every i. By the semantics, this gives us $\mathcal{M}, \mu^a, \mu^e \models \phi^c$.

If ϕ is an existentially quantified disjunction of conjunctions of atoms $\exists \vec{x}_1 . B_1 \vee \cdots \vee \exists \vec{x}_n . B_n$, then $\phi^c = \{B_1^c, \ldots, B_n^c\}$. By assumption, $\mathcal{M}, \mu \models \exists \vec{x}_i . B_i$ for some i, which means that there is an assignment μ' different from μ only in the values assigned to $\vec{x}_i$ such that $\mathcal{M}, \mu' \models B_i$. The difference between μ and μ' is in their existential variables, so we can split μ' into μ^a and μ'^e. By induction, we then have that $\mathcal{M}, \mu^a, \mu'^e \models B_i^c$ for some i, which means that $\mathcal{M}, \mu^a, \mu'^e \models \phi^c$.

Finally, if ϕ is an implication $\forall \vec{x}(A \rightarrow B)$, then $\phi^c = A^c \cup B^c$. By assumption, there is an assignment μ' different from μ only in the values assigned to $\vec{x}$ such that either $\mathcal{M}, \mu' \not\models A$ or $\mathcal{M}, \mu' \models B$. We can split μ' into a universal and an existential part as usual to get μ'^a and μ^e. If $\mathcal{M}, \mu' \not\models A$ holds, then by induction $\mathcal{M}, \mu'^a, \mu^e \not\models A^c$, which means that $\mathcal{M}, \mu'^a, \mu^e \models A^c \cup B^c = \phi^c$. On the other hand, if $\mathcal{M}, \mu' \models B$ was the case, then by induction $\mathcal{M}, \mu'^a, \mu^e \models B^c$ and thus also ϕ^c.

As μ was arbitrarily chosen, this holds for every μ, which means that it is also true that for every universal assignment μ^a there exists an existential assignment μ^e such that $\mathcal{M}, \mu^a, \mu^e \models \phi^c$, which again means that $\mathcal{M} \models \phi^c$. ■

3.2.2 Substitutions and relations

Some care needs to be taken concerning the two types of variables when defining substitutions, unifiers, variants, etc.

Definition 3.17 (Substitutions) *A* substitution *is a function from the set of variables to the set of terms. As we have two types of variables, we split substitutions into a universal and an existential part.*

A universal substitution *is a function $\sigma^a : \mathcal{V}^a \to \mathcal{T}$ from the set of universal variables to the set of terms.*

An existential substitution *is a function $\sigma^e : \mathcal{V}^e \to \mathcal{T}$ from the set of existential variables to the set of terms.*

A faithful variable renaming *$\dot{\sigma}$ is a bijective substitution that maps existential to existential and universal to universal variables.*

A refreshing substitution *(refresher) ϱ for a clause or a literal is a faithful variable renaming that maps every occurring (universal and existential) variable to a fresh variable with respect to a tableau branch, which will always be clear from the context.*

A witnessing substitution *is an injection $\upsilon : \mathcal{V}^e \to \mathcal{W}$ that maps every existential variable to a witness. This makes a witnessing substitution an existential substitution.*

A universal grounding substitution *γ^a for a clause C is a substitution that maps every universal variable in C to a ground term. An* existential grounding substitution *γ^e for C likewise maps every existential variable in C to a ground term.* □

Example 3.18 *Consider the universal substitution $\sigma = \{x/a, y/\dot{u}\}$ and the existential substitutions $\tau = \{\dot{u}/x, \dot{v}/c\}$ and $\delta = \{\dot{u}/\dot{w}, \dot{x}/\dot{y}, z/w\}$. We have that σ maps the variable x to the constant a and the variable y to the existential variable $\dot{u}$. The other two substitutions are read the same way.*

The substitution δ is a faithful variable renaming, as it is bijective and maps existential variables to other existential variables. On the other hand, τ is not a renaming, as it maps a variable to a constant.

The substitution δ is a refresher for any branch which does not contain the variables $\dot{w}$, $\dot{y}$, and w.

Given the clause $C = Px \vee Q\dot{v}$, the substitution σ is a universal grounding substitution for C, as $C\sigma = Pa \vee Q\dot{v}$ contains no universal variables. However, σ is not a most general grounding substitution, as $\{x/a\}$ is more general than σ. Likewise, τ is an existential grounding substitution for C, as $C\tau = Px \vee Qc$ contains no existential variables. □

Definition 3.19 (Unifier, multiset unifier) *A* unifier *for clauses C and D is a substitution σ such that $C\sigma = D\sigma$.*

A multiset unifier *for two sets of literals $S = \{s_1, \ldots, s_n\}$ and T is a substitution σ such that $S\sigma = T\sigma$, where $S\sigma = \{s_1\sigma, \ldots, s_n\sigma\}$.*

In this thesis we will use universal unifiers*, and we will explicitly state this every time we use one.* □

Example 3.20 *Given the clauses $C = Pxc \vee Qa$ and $D = P\dot{u}z \vee Qy$, the substitution $\sigma = \{x/\dot{u}, z/c, y/a\}$ is a unifier for C and D, as $C\sigma = D\sigma = P\dot{u} \vee Qa$. This unifier is universal, as it does not contain existential variables in the domain.*

Given two sets of literals $S = \{Pa, Px, Qb\}$ and $T = \{Pa, Qy\}$, the universal substitution $\tau^a = \{x/a, y/b\}$ is a universal unifier for S and T, as $S\tau^a = T\tau^a = \{Pa, Qb\}$. □

Definition 3.21 (Limited universal substitution and relations) *A* limited universal substitution *λ^a is a universal substitution such that, for any universal variable x and witnessing substitution υ, $x\lambda^a\upsilon = x\lambda^a$. In other words, a limited universal substitution cannot introduce an existential variable into a clause.*

For clauses A and B we define $A \geq_0 B$, true iff there is a limited universal substitution λ^a and existential faithful variable renaming $\dot{\sigma}$ such that $A\lambda^a\dot{\sigma} = B$. We also define $A >_0 B$, true iff $A \geq_0 B$ and not $B \geq_0 A$. □

Example 3.22 *The universal substitution $\sigma^a = \{x/c, y/z\}$ is limited, while the universal substitution $\tau^a = \{x/\dot{u}, y/c\}$ is not, as $x\tau^a\upsilon = \bar{u} \neq x\tau^a = \dot{u}$.* □

Example 3.23 *Consider the clauses $Px\dot{u}z$, $Py\dot{v}z$ and $Pa\dot{u}\bar{w}$. The following holds: $Px\dot{u}z \geq_0 Py\dot{v}z >_0 Pa\dot{u}\bar{w}$. The first part of this inequality holds because we can use limited universal substitution $\lambda^a = \{x/y\}$ and existential faithful variable renaming $\dot{\sigma} = \{\dot{u}/\dot{v}\}$ to get $Px\dot{u}z\lambda^a\dot{\sigma} = Py\dot{v}z$. The second part holds using limited universal substitution $\kappa^a = \{y/a, z/\bar{w}\}$ and the inverse of $\dot{\sigma}$.*

On the other hand, we have that $Px \geq_0 P\dot{u}$ does not hold, as there is no limited *universal substitution that can send x to $\dot{u}$. Also, since $Py\dot{v} \geq_0 Px\dot{u}$ holds, $Px\dot{u} >_0 Py\dot{v}$ does not hold.* □

Definition 3.24 (Relations) *For clauses A and B we define $A \succ B$, A is* more general *than B, iff there exists a limited universal substitution σ^a and witnessing substitution υ such that $A\sigma^a\upsilon = B$.*

A and B are faithful variants*, denoted $A \sim_f B$, iff there exists a faithful variable renaming $\dot{\sigma}$ such that $A\dot{\sigma} = B$.* □

Example 3.25 *Consider the clauses $P\dot{u}z$, $P\dot{w}v$, and $P\bar{x}y$. The following holds:*

$$P\dot{u}z \sim_f P\dot{w}v \succ P\bar{x}y \succ P\bar{x}y$$

The relation $P\dot{u}z \sim_f P\dot{w}v$ holds because we can use a faithful variable renaming $\dot{\sigma} = \{\dot{u}/\dot{w}, z/v\}$ to get $P\dot{u}z\dot{\sigma} = P\dot{w}v$. The relation $P\dot{w}v \succ P\bar{x}y$ holds because we can use the limited universal substitution $\sigma^a = \{v/y\}$ and some witnessing substitution υ such that $\dot{w}\upsilon = \bar{x}$ to get $P\dot{w}v\sigma^a\upsilon = P\bar{x}y$.

Of special interest is $P\bar{x}y \succ P\bar{x}y$, as the symbol $\succ$ does not suggest this. It holds because we can use the empty limited universal substitution $\emptyset$ and any witnessing substitution (as there are no existential variables, it has no effect) to satisfy Def. 3.24. However, $\succ$ is not *reflexive; cf. Example 3.26.*

Going in the other direction, we have that $P\bar{x}y \prec P\bar{x}y$ holds (for obvious reasons), but that $P\dot{w}v \prec P\bar{x}y$ does not hold, as there is no way to map $\bar{x}$ to anything. □

Example 3.26 *As a cautionary example, $P\dot{u}\dot{v} \succ P\dot{u}\bar{v}$ does not hold, as a witnessing substitution is total on existential variables. This means that $P\dot{u} \succ P\dot{u}$ does not hold either, as there is no witnessing substitution υ such that $\dot{u}\upsilon = \dot{u}$. However, $Px \succ Px$ does hold, as an empty substitution counts as a limited universal one.*

Also, $P\dot{u} \succ P\dot{v}$ does not hold, as neither a universal nor a witnessing substitution may map $\dot{u}$ to $\dot{v}$. Finally, $Px \succ P\dot{u}$ does not hold either, as no limited universal substitution may introduce existential variables. These properties may be counterintuitive, but make some of the proofs nicer. Taken together, we can say that the $\succ$ relation is useful for comparing clauses without witnesses to clauses with witnesses (but without existential variables), and also to compare clauses with witnesses to each other. For comparisons between clauses with existential variables we shall use the relations from Def. 3.21. □

Chapter 4

The calculus

This chapter is organized as follows: In Section 4.1, we go through an example derivation to explain the usage of our calculus. In Section 4.2, we give the technical definitions necessary to formally define the rules, which we do in Section 4.2.1. Then, we discuss the conditions found in the rules in Section 4.3, focusing on possible versus impossible rule applications. The chapter is then concluded by a discussion of problems encountered on the way and the differences between our work and that of Baumgartner [Bau98] in Section 4.4.

4.1 Example and usage

Consider the following unsatisfiable clause set.

(A) $P\dot{u}x$

(B) $Pxy \to Qxy \wedge Rxy$

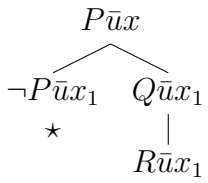
(C) $Qxa \wedge Rxa \to \bot$

We shall derive a closed tableau for it step by step. At the start, we have an empty tableau with a single empty branch, and the negative literals of (A) (of which there aren't any) unify with the empty tableau without instantiating any literals on the branch. This means that we can apply the Ext rule using clause (A) and get the tableau below. The existential variable becomes a witness during this application.

$$P\bar{u}x$$

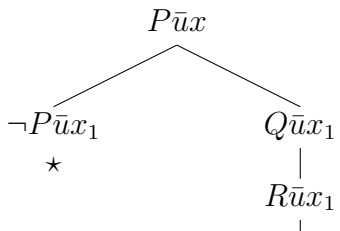
Next, we have clause (B) with negative literals that unify with $P\bar{u}x$. We refresh the variables in the branch literals to get $P\bar{u}x_1$, and apply Ext

using clause (B) and universal substitution $\{x/\bar{u}, y/x_1\}$ to get the tableau below. The left-hand branch of this tableau is defined to be *closed* by Def. 4.4, and we mark such branches by a $\star$.



Next, we would like to close the single open branch by applying Ext with (C), as the negative literals present there unify with the branch literals. However, the unifier computed here is different from the two unifiers we computed previously. If we take fresh copies $\{Q\bar{u}x_2, R\bar{u}x_3\}$ of $Q\bar{u}x_1$ and $R\bar{u}x_1$, the universal MGU for this set and the negative literals of (C) is $\{x/\bar{u}, x_2/a, x_3/a\}$, a unifier that instantiates the branch literals. This renders the Ext rule inapplicable, but allows us to use the Link rule. This rule operates on the *current clause set* of a branch, which is a set of clauses that can be used in Ext rule applications on that branch. The calculus maintains one such set for every branch, and the Link rule adds instances of existing clauses to this set.

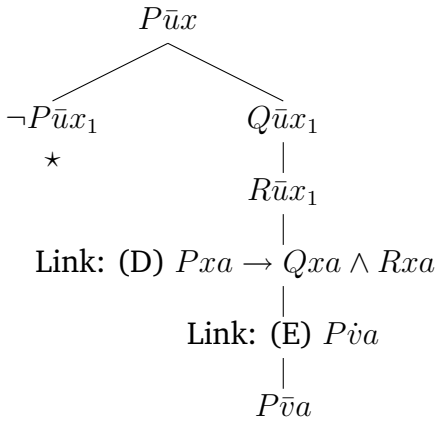
The result of our Link rule application can be seen in the tableau below. The rule is applied with clause (C) not using the branch literals, but the corresponding literals in the *origin clauses*, i.e. clauses from the current clause set that were used in Ext applications on the branch we are currently working on. In this case, this is clause (B), but twice, as we are using two literals from it. We refresh the variables in one of the instances to get $Px_2y_2 \to Qx_2y_2 \wedge Rx_2y_2$ (for the other instance, we shall use index 3). Then, we unify $\{Qx_2y_2, Rx_3y_3\}$ with the negative literals of (C). The resulting two copies of $Pxa \to Qxa \wedge Rxa$ are faithful variants of each other (identical copies, in fact), so it suffices to add only one of them to the current clause set.



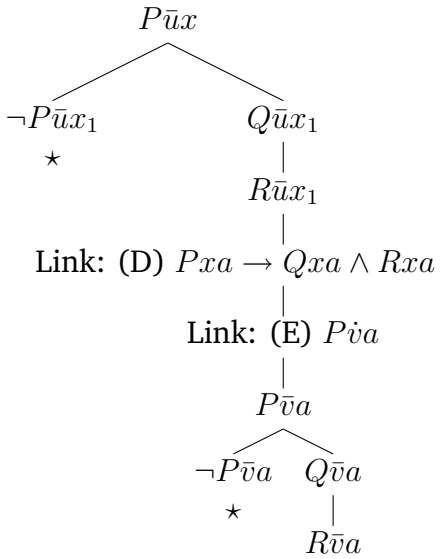
Link: (D) $Pxa \to Qxa \wedge Rxa$

After this, we still cannot use (C) in an Ext rule application, as there has been no change to the branch literals yet. Nor can we use (D) with

Ext, as the negative literals of (D) unify with $P\bar{u}x$ (modulo refreshers) only by instantiating the branch literal. However, this allows us to use Link again to generate the tableau below. The Link application with (D) adds (E) to the current clause set, and as (E) has no negative literals we can use Ext on it directly, also shown on the tableau below. We introduce a *new* witness for the existential variable when we do this.

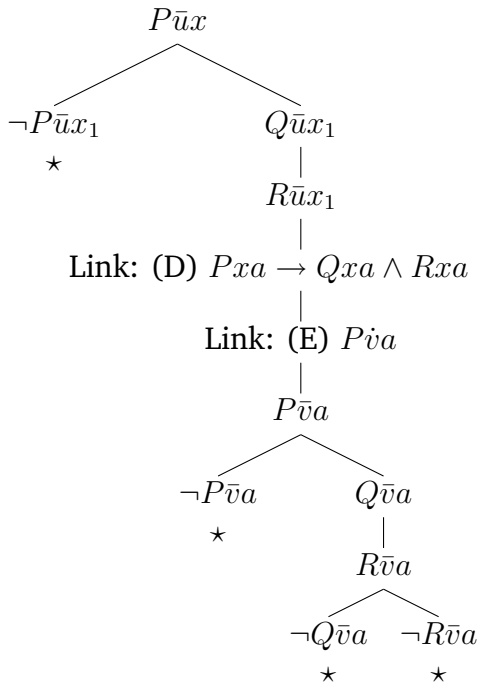


Now (D) can be used in an Ext application, as Pxa unifies with $P\bar{v}a$ using MGU $\{x/a\}$. The result of this application is shown in the tableau below, and a branch is again closed.



Now we can apply Ext with (C) using unifier $\{x/\bar{v}\}$ on branch literals $Q\bar{v}a$ and $R\bar{v}a$. This application yields the complete, closed tableau below. Note that it is not necessarily the case that we end up with ground

instances of the original clauses on the tableau — a refutation may be, and often is, found before we reach ground level. The main point of the calculus remains: We try to unify negative literals with branch literals. If the branch literals are instantiated by this, we apply Link, otherwise we apply Ext.



In this derivation, we applied Ext until we hit an instantiation problem, then used Link once. This did not allow us to proceed, as (D) could not be used with Ext directly — however, we used Link to keep instantiating up the tableau, a process that later allowed us to use Ext again. This is typical for this calculus, and also for the NG calculus of Baumgartner. We expect to be able to make use of this observation in an implementation, by keeping track of the clauses that need to be instantiated with Link in the case of a failed Ext, and thereby finding necessary Link applications very quickly. A formal treatment of this effect can be found in Chapter 7.

4.2 Formal definitions

Before we define the rules of our calculus, we give some necessary technical definitions.

Definition 4.1 (Tableau, branch) *A* tableau *is a set of branches.*

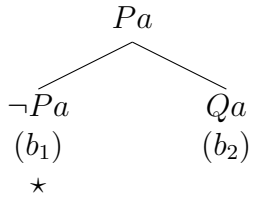
A branch b *is a tuple* $\langle p, N, TC, OR\rangle$*, where* p *is a list of literals (the* branch list*),* N *is the* current clause set *of the branch,* TC *is a relation between the literals in* p *and the set of* tableau clauses *of the branch, and* OR *is a relation between the literals of* p *and the set of* origin clauses *of the branch. (We will define what all four of these contain during the description of the calculus.) To talk about the elements of a branch, we define suitable notation as follows:*

1. *A function* ℓ *such that* $\ell(b) = p$*,*
2. *another function* C^- *such that* $C^-(b) = N$*,*
3. *a relation* TC_b *such that* $TC_b(L, C)$ *if* $\langle L, C\rangle \in TC$*, and*
4. *a relation* OR_b *such that* $OR_b(L, C)$ *if* $\langle L, C\rangle \in OR$*.*

We overload $\in$ *and use it to denote list membership:* $L \in p$ *means that* L *appears in the list* p*, possibly several times. A list is written using square brackets; the empty list is thus* $[]$*.*

We write $p \cdot L$ *for the extension of the list* p *with a new element* L *or, if* L *is a set, with the elements of* L *in any order.* □

Example 4.2 *Consider the clause set* $S = \{Pa, Px \to Qx\}$ *together with the tableau drawn below:*



The tableau contains two branches

$$b_1 = \left\langle \begin{array}{lcl} p_1 & = & [Pa, \neg Pa] \\ N_1 & = & \{Pa, Px \to Qx\} \\ TC_1 & = & \{\langle Pa, Pa\rangle, \langle \neg Pa, Pa \to Qa\rangle\} \\ OR_1 & = & \{\langle Pa, Pa\rangle, \langle \neg Pa, Px \to Qx\rangle\} \end{array} \right\rangle$$

and

$$b_2 = \left\langle \begin{array}{lcl} p_2 & = & [Pa, Qa] \\ N_2 & = & \{Pa, Px \to Qx\} \\ TC_2 & = & \{\langle Pa, Pa\rangle, \langle Qa, Pa \to Qa\rangle\} \\ OR_2 & = & \{\langle Pa, Pa\rangle, \langle Qa, Px \to Qx\rangle\} \end{array} \right\rangle$$

For p_1, we have that $Pa, \neg Pa \in p_1$, and that $p_1 \cdot Qc = [Pa, \neg Pa, Qc]$. The current clause sets of the two branches are equal, but the relations TC and OR differ. Why they differ is explained in Section 4.2.1. □

Definition 4.3 (Fresh witnessing substitution) *We say that a witnessing substitution υ is* fresh *with respect to a branch b if no witness on b is in the co-domain of υ. A witnessing substitution is fresh with respect to a tableau T if it is fresh with respect to every $b \in T$.* □

Given these definitions, we can now formally define the rules of our calculus.

4.2.1 Rules

The rules are formally defined below. To allow the Link rule to refer to the original clauses, Ext keeps references to them using the relations TC and OR. Both rules operate on tableaux, and the process is started by constructing an initial tableau $\{\langle [], C^s, \{\}, \{\}\rangle\}$ from a clause set C^s.

Definition 4.4 (The Ext rule) *The Ext rule schema is*

$$\frac{\langle p, N \cup \{A \to B\}, TC, OR\rangle, T}{\{\langle p \cdot K, N \cup \{A \to B\}, TC'_K, OR'_K\rangle \mid K \in Elems((A \to B)\sigma^a\upsilon)\} \cup T}\ \textit{Ext}$$

where

1. *$\langle p, N \cup \{A \to B\}, TC, OR\rangle$ is a branch,*
2. *T contains the remaining branches*
3. *σ^a is a universal substitution, and*
4. *υ is a fresh witnessing substitution w.r.t. the current tableau*

such that

5. *$L_1, \ldots, L_n \in p$ are literals with refreshers $\varrho_1, \ldots, \varrho_n$,*
6. *σ^a is a multiset MGU such that $A\sigma^a = \{L_1\varrho_1, \ldots, L_n\varrho_n\}\sigma^a$, and*
7. *for every i, $L_i\varrho_i\sigma^a \sim_f L_i\varrho_i$.*

Additionally,

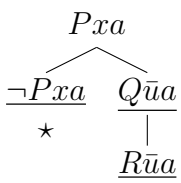
8. *every new branch* $\langle p \cdot \neg K, \ldots \rangle$*, where* $K \in A\sigma^a\upsilon$*, is called* closed *(we mark closed branches by a* $\star$*),*
9. *every new branch* $\langle p \cdot K, \ldots \rangle$*, where* $K \in B\sigma^a\upsilon$*, is called* open*,*
10. $TC'_K = TC \cup \{\langle K, (A \to B)\sigma^a\upsilon\rangle\}$*, and*
11. $OR'_K = OR \cup \{\langle K, A \to B\rangle\}$. □

In other words, if we can unify the negative literals of a clause with the literals on the branch without instantiating said literals, we split the clause below the branch and introduce witnesses for any existential variables. The relations TC and OR are updated to relate the literals added on the new branches to the correct tableau clause and origin clause.

Given a literal on a branch, we thus know the tableau clause it belongs to (TC) and what clause in the current clause set this clause came from (OR). This is illustrated in the figure below: The underlined literals in the derivation form a tableau clause, namely $Pxa \to Q\bar{u}a \wedge R\bar{u}a$, and every underlined literal is TC-related to it. Also, every underlined literal has (B) as an origin clause, and is related to it by OR.

(A) Pxa

(B) $Pxy \to Q\dot{u}y \wedge R\dot{u}y$



On a separate note, observe that the MGU σ^a above is in fact limited, since neither A nor the literals on the branch can ever contain existential variables. It therefore holds that $(A \to B) \succ (A \to B)\sigma^a\upsilon$.

Definition 4.5 (The Link rule) *The Link rule schema is*

$$\frac{\langle p, N \cup \{A \to B\}, TC, OR\rangle, T}{\langle p, N \cup \{A \to B\} \cup \{C_1\varrho_1\sigma^a, \ldots, C_n\varrho_n\sigma^a\}, TC, OR\rangle, T}\ \textit{Link}$$

where

1. $\langle p, N \cup \{A \to B\}, TC, OR\rangle$ *is a branch,*
2. T *contains the remaining branches, and*
3. σ^a *is a universal substitution*

such that

4. *$L_1, \ldots, L_n$ are literals from clauses $C_1, \ldots, C_n \in N$ with refreshers $\varrho_1 \ldots, \varrho_n$, such that $OR(L'_i, C_i)$ for some $L'_i \in p$ with $L_i \succ L'_i$ for every i, and*

5. *σ^a is a multiset MGU such that $A\sigma^a = \{L_1\varrho_1, \ldots, L_n\varrho_n\}\sigma^a$, and*

6. *$L_i\varrho_i\sigma^a \not\sim_f L_i\varrho_i$ for some i.*

Instead of adding $\{C_1\varrho_1\sigma^a, \ldots, C_n\varrho_n\sigma^a\}$ to the current clause set, it is sufficient to add only those $C_i\varrho_i\sigma^a$ that satisfy $C_i\varrho_i\sigma^a \not\sim_f C_i\varrho_i$. □

In contrast to the Link rule in [Bau98], we do not unify branch literals with A — instead, we unify the corresponding literals in the origin clauses with A. If we can do this in such a way that the literals are instantiated, we add their clauses to the current clause set.

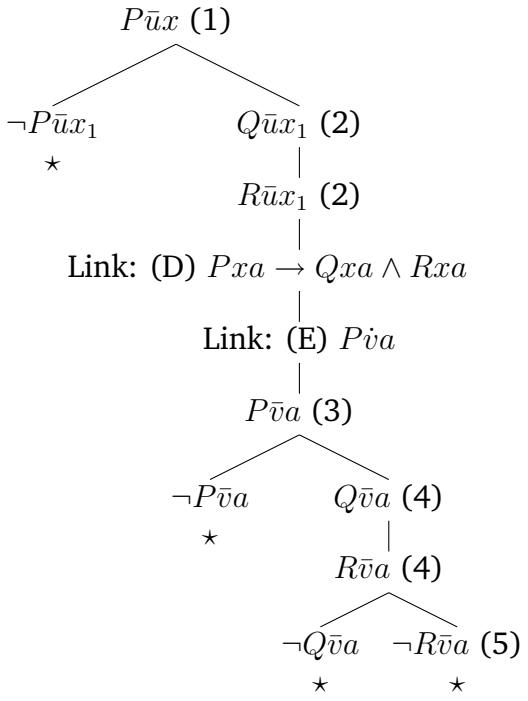
4.3 Possible and impossible rule applications

In this section we shall discuss the conditions found in the rules and focus on what would make a rule application impossible. Consider the example derivation from Section 4.1, repeated here with every Ext rule application marked.

(A) $P\dot{u}x$

(B) $Pxy \to Qxy \wedge Rxy$

(C) $Qxa \wedge Rxa \to \bot$



There are five applications of Ext, two of which put two literals on the same branch, and two applications of Link. We shall discuss some of them in detail.

4.3.1 Ext application 2

Consider the Ext application labeled (2): We take clause (B) from the current clause set, and need to satisfy all the conditions in Def. 4.4. Our branch contains literal $P\bar{u}x$, and we can use refresher $\varrho = \{x/x_1\}$ to satisfy condition 5. The point of condition 5 is to avoid variable conflicts when speaking about the satisfiability of a branch (cf. Def. 5.1). This is accomplished by demanding that the negative literals of a clause be unified with fresh copies of literals on the branch. This condition cannot, in a sense, be violated, as it is always possible to take fresh copies of all the variables present.

Next, we need to check that the literals on the branch (in this case, $P\bar{u}x$) unify with the negative literals of (B) using a universal substitution. The fact that the substitution is universal is explicated, but vacuous: as there are no existential variables in the negative literals of any clause nor in the branch literals of any branch, if there exists a unifier for the two sets of literals, there also exists a universal such unifier. We also demand in condition 6 that the unifier be a MGU. In our case, this unifier is $\sigma^a = \{x/\bar{u}, y/x_1\}$, but there may always exist several MGUs. This condition may, and will, often fail — if (B) had been a clause of the form $Pay \to \cdots$, an Ext rule application would be impossible at this stage, for there is no unifier for $\{P\bar{u}x_1\}$ and $\{Pay\}$.

The most important condition is condition 7, which demands that for every fresh copy of a branch literal L used in the rule application, it must hold that $L\sigma^a \sim_f L$. Recall from Def. 3.24 that $A \sim_f B$ holds if there is a faithful variable renaming $\dot{\sigma}$ such that $A\dot{\sigma} = B$. The point of this is as follows: The unifier σ^a must not instantiate any of the branch literals, i.e. map any variable to a term that cannot be mapped back to that variable. In our case, this is true: $P\bar{u}x_1\sigma^a = P\bar{u}x_1 \sim_f P\bar{u}x_1$. This condition can fail, and in surprising ways.

Assume that (B) was a clause of the form $Pxy \wedge Pyz \to \cdots$. In this case, no Ext application is possible, because there is no unifier for $\{P\bar{u}x_1, P\bar{u}x_2\}$ and $\{Pxy, Pyz\}$ that avoids mapping x_1 to $\bar{u}$. Notice that we may try (and even succeed) applying Ext on a branch with only one literal using a clause with several negative literals. We can do so by taking copies of the branch literals, each with its own refresher.

Another interesting fact about condition 7 is that if it alone fails, then

a Link application is possible instead of Ext. The proof of this property is in Lemma 6.5, and it suggests a search strategy: Apply Ext whenever possible, and if the application fails due to condition 7, apply Link instead. This application of Link will sooner or later lead you to an Ext application — cf. Chapter 7 for a detailed overview of the interplay between Ext and Link.

There is also a witnessing substitution υ mentioned in condition 4. It is applied to the clause (B) after σ^a to replace every existential variable by a witness, and this substitution must be *fresh*, i.e. it must not introduce witnesses already present. However, as it is a substitution, it will map the same existential variable in (B) to the same witness, as intended. This is also a condition that cannot be violated, since we have an infinite supply of witnesses. In our example this condition is not needed, as (B) contains no existential variables.

Finally, Def. 4.4 has an 'additionally' part. This is bookkeeping information. The first two items here state that any new branch with a negative literal is closed — this is generally true, and so is the fact that any branch with no negative literal on it is open. The last two items state that we keep track, for every literal added by any Ext application, of the clause this literal came from (TC) and where that clause came from (OR). For example, the branch with literal $Q\bar{u}x_1$ would get a pair $\langle Q\bar{u}x_1, P\bar{u}x_1 \to Q\bar{u}x_1\rangle$ added to its TC relation, and a pair $\langle Q\bar{u}x_1, Pxy \to Qxy\rangle$ added to its OR relation.

4.3.2 Link applications

Consider the first Link application in the derivation above, the one labeled (D). It uses clause (C) from the current clause set, but before explaining it, let us recall Def. 4.5. Here, we find condition 4, which demands that we have literals $L_1, \ldots, L_n$ from clauses $C_1, \ldots, C_n$ in the current clause set (in our case, this is the set of clauses $\{A, B, C\}$). These literals and clauses must be such that for every i, there exists a literal L'_i on the branch with a pair $\langle L'_i, C_i\rangle$ in the relation OR of that branch, and furthermore $L'_i \succ L_i$ must hold.

Let us turn this definition around: There must exist literals $L'_1, \ldots, L'_n$ on the branch such that there are literals $L_1, \ldots, L_n$ in their origin clauses (clauses related to the literals by means of OR) with $L'_i \succ L_i$ for every i. In our case, there are literals $Q\bar{u}x_1, R\bar{u}x_1$ on the branch, both with origin clause (B), and (B) contains literals Qxy, Rxy, both of which are more general than their branch counterparts. Notice that these literals come from two distinct copies of (B). We refresh the literals to get Qx_2y_2, Rx_3y_3 from $Px_2y_2 \to Qx_2y_2 \vee Rx_2y_2$ and $Px_3y_3 \to Qx_3y_3 \vee Rx_3y_3$ respectively,

and move on to condition 5.

Condition 5 is similar to condition 6 in the Ext rule, and demands that there exists a universal unifier σ^a for the set $\{Qx_2y_2, Rx_3y_3\}$ and the negative literals of (C), namely $\{Qxa, Rxa\}$. Here, the fact that the unifier is universal is important — the literals from the origin clauses may contain existential variables, and we do not wish to instantiate them with 'ordinary' terms under any circumstances. For example, the literals $Pa\dot{w}$ and Pxb unify, but there is no universal unifier for them — so this condition may fail in two different ways, in contrast to condition 6 in the Ext rule, where the 'universal' part is vacuous.

In our case, there is a universal unifier $\sigma^a = \{x_2/x, y_2/a, x_3/x, y_3/a\}$ which is a MGU as demanded by condition 5, and we may move on to condition 6. This condition is the opposite of condition 7 in the Ext rule, and states that at least one of the origin clauses that we are working on must be instantiated by σ^a. In our case, this is true for both clauses, but as the resulting clauses are variants (in fact, exact copies) of each other, we need only add one of them to the current clause set. As a result, we add the clause $Pxa \to Qxa \vee Rxa$ to the current clause set. Notice that this condition may fail — in that case, the rule application is impossible because we would add variants of clauses already present to the current clause set, which is unnecessary.

4.4 Discussion

Our calculus is an adaptation of the Next-Generation Hyper Tableaux calculus (NG) [Bau98], and has two rules called Ext and Link. The purpose of the Ext rule, here as in NG, is to split a clause below a branch, possibly resulting in new branches, if the left-hand side of this clause unifies with the literals present on the branch. However, this is only done if the unifier does not instantiate the literals on the branch. If existential variables are present in the clause to be split, the Ext rule replaces them by fresh constants.

For example, given the clause $P\dot{u}x$, which corresponds to the first-order formula $\forall x \exists u Pux$, one extends the tableau with the literal $P\bar{u}x$ for a new witness $\bar{u}$.

The Link rule is, on the other hand, significantly different from the one in NG. Its purpose in NG is to handle the case where the unifier computed instantiates branch literals, by adding the instances we would get from this unifier to the current clause set. Here, the purpose remains, but matters are complicated by the presence of existential variables and the

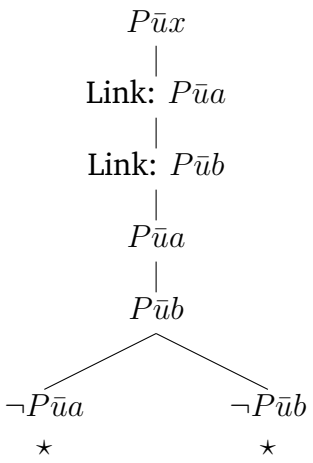
constants that replace them. The first complication is to preserve witness freshness when instantiating. Consider an example to illustrate the necessity of this (the examples here assume a familiarity with NG, but this will not be needed later):

Clause set:

(A) $P\dot{u}x$

(B) $Pya \wedge Pyb \rightarrow \bot$

Derivation:



The clause set is satisfiable, but we get a closed tableau. The obvious solution to this is to make Link substitute new witnesses for the ones present. As a Link rule application necessarily instantiates the clauses, this seems like a good idea. However, if this is done carelessly it leads to a completeness problem, as shown below.

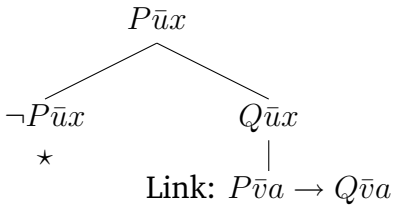
Clause set:

(A) $P\dot{u}x$

(B) $Pyz \rightarrow Qyz$

(C) $Qwa \rightarrow \bot$

Derivation:



A Link rule application with clause (C) on the right branch leads to an instantiation of the tableau clause $P\bar{u}x \rightarrow Q\bar{u}x$ with x/a. A new witness $\bar{v}$ is generated, so the Link application results in $P\bar{v}a \rightarrow Q\bar{v}a$. Unfortunately, we lose completeness, because the new witness makes it impossible to Link $P\bar{u}x$ with $P\bar{v}a \rightarrow Q\bar{v}a$. The point here is that the interpretation of $\bar{u}$ does not depend on the value of y at all, and therefore it is sound to keep $\bar{u}$ unchanged when instantiating y. This motivates our alteration of the Link rule to create instances of the original, uninstantiated clauses

in the current clause set, instead of the tableau clauses. This removes all problems of preserving freshness, as the current clause set contains existential variables, and these become constants only in the Ext rule.

This version of the Link rule raises an interesting question: How do you efficiently find possible applications of Link? For Ext, you need only to look at the branch literals and unify them with the negative part of a clause. The Link rule, however, is not applied to tableau clauses (of which the branch literals are part), it is applied to origin clauses. The answer is given in Lemma 6.5, where we prove that it is sufficient to look at the branch literals — if the negative part of a clause unifies with the branch literals in such a way that the literals are instantiated, then it is possible to apply Link. This allows us to 'have our cake and eat it too', as we manage to handle existential variables without losing efficiency.

Chapter 5

Soundness

We prove soundness by showing that the rules of our calculus preserve satisfiability of tableaux, while a tableau with all branches closed is not satisfiable. This is the usual way of proving soundness for tableau and sequent systems. The tricky part here is soundness of the Ext rule — the Link rule generates instances of the clauses we started with, and any instance of a clause is a logical consequence of it, hence it is sound to add it to the current clause set. Soundness of Ext is similar to soundness of Ext in [Bau98], but there only an informal proof is given. We therefore write the proof out in full for both rules.

Definition 5.1 (Branch satisfiability) *We extend the notion of satisfiability defined in Section 3.2.1 to branches. A model $\mathcal{M}$ satisfies a branch b under universal assignment μ^a if*

1. *$\mathcal{M}$ satisfies every clause in $C^-(b)$, and*
2. *$\mathcal{M}, \mu^a$ satisfies every literal in $\ell(b)$.*

Notice that there is no mention of existential assignment. This is so because the literals on a branch do not contain existential variables. The first condition does not refer to the assignments at all, since the variables in the clauses are implicitly quantified according to their type. □

Example 5.2 *Let b be a branch with branch list $\ell(b) = [Q\bar{v}z, P\bar{v}y]$ and current clause set $C^-(b) = \{Q\dot{v}z, Qxy \rightarrow Pxy, Rc\}$. The model $\mathcal{M}$ with $|\mathcal{M}| = \{\bar{v}, a, c\}$ and satisfying only the ground atoms*

$$Q\bar{v}\bar{v}, Q\bar{v}a, Qac, P\bar{v}a, P\bar{v}\bar{v}, Pac, Rc$$

satisfies b under universal assignment $\mu^a = \{z/\bar{v}, y/a\}$.

However, $\mathcal{M}$ does not satisfy b under the universal assignment $\eta^a = \{z/c, y/a\}$ because $\eta^a(z) = c$ and $\mathcal{M} \not\models Q\bar{v}c$. □

Lemma 5.3 (Refreshing substitutions preserve satisfiability) *For every clause C, model $\mathcal{M}$, and refresher ϱ, $\mathcal{M} \models C$ if and only if $\mathcal{M} \models C\varrho$.* □

Proof *Let a clause C, model $\mathcal{M}$, and refresher ϱ be given. For the only if part, assume that $\mathcal{M} \models C$, and let η^a be an arbitrary universal assignment for $\mathcal{M}$. Define a universal assignment μ^a for $\mathcal{M}$ such that $\mu^a(x) = \eta^a(x\varrho)$ for every universal variable x. (Recall from Def. 3.17 that for any variable x, $x\varrho$ is a variable of the same type as x.)*

Since $\mathcal{M}$ satisfies C for every universal assignment by assumption, it must in particular satisfy C under μ^a, which means that there exists an existential assignment μ^e such that $\mathcal{M}, \mu^a, \mu^e \models C$.

Now construct an existential assignment η^e by letting $\eta^e(\dot{u}\varrho) = \mu^e(\dot{u})$ for every existential variable $\dot{u}$. Since $\mathcal{M}, \mu^a, \mu^e \models C$, it must also be the case that $\mathcal{M}, \eta^a, \eta^e \models C\varrho$. Since η^a was arbitrary, this holds for every universal assignment, and we are done.

The same proof goes through for the if part if one lets C and $C\varrho$ switch places. ■

Lemma 5.4 *Let $e \in |\mathcal{M}|$ be an arbitrary element of some model. No closed branch b is satisfiable under the universal assignment $*$ such that $*(x) = e$ for every universal variable x.* □

Proof *Assume for the sake of contradiction that there exists a closed branch b and a model $\mathcal{M}$ such that $\mathcal{M}, *$ satisfies b. From the definition of satisfiability, we have that for every $L \in \ell(b)$, $\mathcal{M}, * \models L$. Furthermore, since b is closed, it was closed by an application of the Ext rule, which means that there is a literal $\neg K \in \ell(b)$.*

From the definition of Ext, we know that there exists a universal substitution σ^a, a literal $L \in \ell(b)$, a refresher ϱ, and a witnessing substitution υ such that $K = K'\sigma^a\upsilon = L\varrho\sigma^a$. In other words, K came from some literal K' in a clause using σ^a, and the substitution made it equal to some literal L previously on the branch (ignoring the refresher for a moment).

Since $\mathcal{M}, * \models L$ and L contains no existential variables, $\mathcal{M}, * \models L\varrho$. (By definition of a refresher, every universal variable in L is still universal, and $*$ interprets all universal variables as the same element.) Also, since the Ext rule demands that $L\varrho\sigma^a \sim_f L\varrho$, and using the definition of $\sim_f$, we know that $\mathcal{M}, * \models L\varrho\sigma^a$.

This, however, is a contradiction, since $K = L\varrho\sigma^a$, and a model cannot satisfy a literal and its negation under the same assignment. ■

Before continuing, we shall need the definition below.

Definition 5.5 (Satisfiable tableau) *A model $\mathcal{M}$ satisfies a tableau T if there exists a branch $b \in T$ such that $\mathcal{M}$ satisfies b under the universal assignment $*$ from Lemma 5.4. A tableau is satisfiable if there exists a model that satisfies it.* □

Lemma 5.6 (Ext and Link preserve satisfiability) *If either the Ext or the Link rule is applied on a satisfiable tableau, the resulting tableau is also satisfiable.* □

Proof *Let a satisfiable tableau T be given, and let $\mathcal{M}$ be the model satisfying it. Then, there exists a branch $b \in T$ such that $\mathcal{M}, *$ satisfies b. Notice that if the rule application is not done on b, the resulting tableau still contains b, and is therefore satisfied by $\mathcal{M}$. Assuming that the rule application is indeed done on b, we get a case for each rule.*

Link *An application of the Link rule affects only the current clause set $C^-(b)$, leaving us free to ignore the literals on b. By definition of branch satisfiability, we have that $\mathcal{M} \models C$ for every $C \in C^-(b)$. Furthermore, let $C_1\varrho_1\sigma^a, \ldots, C_n\varrho_n\sigma^a$ be the clauses added to $C^-(b)$ by an application of the Link rule using universal substitution σ^a. By Lemma 5.3, $\mathcal{M} \models C_i\varrho_i$ for every i. Also, since σ^a is a universal substitution (cf. Def. 4.5) it instantiates only the universal variables in $C_i\varrho_i$, from which it follows that $\mathcal{M} \models C_i\varrho_i\sigma^a$ for every i.*

Ext *The Ext rule creates new branches, but copies the current clause set $C^-(b)$ without changing it — we can thus concentrate on the literal list $\ell(b)$. By definition of branch satisfiability, we have that $\mathcal{M} \models C$ for every $C \in C^-(b)$ and that $\mathcal{M}, * \models L$ for every $L \in \ell(b)$. Furthermore, let $b_1, \ldots, b_n$ be the branches added to the tableau by the Ext rule applied on b with clause $A \to B \in C^-(b)$, and denote the resulting tableau by $T' = (T - \{b\}) \cup \{b_1, \ldots, b_n\}$. The clause added to the tableau is $(A \to B)\sigma^a\upsilon$ (cf. Def. 4.4).*

We shall prove that T' is satisfiable by constructing a model $\mathcal{M}'$ satisfying b_i for some i under $*$.

$\mathcal{M}$ satisfies $A \to B$, it also satisfies its instance $(A \to B)\sigma^a$ since σ^a is a universal substitution. In particular this means that for the universal assignment $*$ there is an existential assignment μ^e such that $\mathcal{M}, *, \mu^e \models (A \to B)\sigma^a$. Using this fact, construct $\mathcal{M}'$ by letting $|\mathcal{M}'| = |\mathcal{M}|$. Likewise, for every predicate symbol, constant, or witness x present in T, let $x^{\mathcal{M}'} = x^{\mathcal{M}}$. Finally, for every witness $\dot{u}\upsilon$ in $(A \to B)\sigma^a\upsilon$, let $(\dot{u}\upsilon)^{\mathcal{M}'} = \mu^e(\dot{u})$. As $\dot{u}\upsilon$ is a fresh constant for every $\dot{u}$ in $(A \to B)\sigma^a\upsilon$ (due to the Ext rule, cf. Def. 4.3), $\mathcal{M}'$ is well-defined and satisfies b under $*$. In addition, due to the fact that $\mathcal{M}, *, \mu^e \models (A \to B)\sigma^a$ it is also the case that that $\mathcal{M}', * \models$

$(A \to B)\sigma^a v$. This means that $\mathcal{M}', *$ satisfies some $K \in (A \to B)\sigma^a v$. As we for some i have $\ell(b_i) = \ell(b) \cdot K$, our new model $\mathcal{M}'$ satisfies b_i under $*$ and we are done. ■

Theorem 5.7 (Soundness) *The presented calculus is sound: If there exists a closed tableau derivation for a clause set S, then S is unsatisfiable.* □

Proof *Let S be given, and assume that we have a closed tableau derivation for it. Furthermore assume, for contradiction, that S is satisfiable. By Lemma 5.6, any tableau derivation for S must contain a branch satisfied by some model under the assignment $*$ from Lemma 5.4. However, since all branches are closed, by Lemma 5.4 no such branch exists, a fact that contradicts our assumption about the satisfiability of S.* ■

Chapter 6

Completeness

In this chapter we prove completeness. First, we define a few auxiliary notions and prove Lemma 6.5 to be used in the proof of Lemma 6.10. Then, we define how to construct a model from an open branch in Def. 6.8, prove Lemma 6.10, and finally prove our calculus complete in Theorem 6.11.

Definition 6.1 (Tableau clauses) *Let b be a branch. By the set of* tableau clauses *of or on b, written $C^+(b)$, we mean the set*

$$\bigcup_{L \in \ell(b)} \{C \mid TC_b(L, C)\}$$

Intuitively, this is the set of instantiated clauses $(A \to B)\sigma^a\upsilon$ that were used in Ext applications on b. Likewise, the set of origin clauses *is the set of clauses $A \to B$ above, using relation OR_b.* □

Definition 6.2 (Derivation) *A* tableau derivation *for a clause set S is a sequence of tableaux $P_0, P_1, \ldots$, possibly infinite, such that $P_0 = \{\langle [], S, \{\}, \{\}\rangle\}$ and any P_{n+1} is obtained from P_n by an application of a rule to a branch of P_n.* □

Definition 6.3 (Refutation) *A* refutation *of a clause set S, also called a* proof*, is a tableau derivation for S that contains at least one tableau with all branches closed.* □

Definition 6.4 (Finished branch, fair derivation) *A branch b is* finished *if all the rules that can be applied have been applied.*

A derivation is fair *if it is a refutation or if it contains a tableau with a finished branch.* □

When searching for ways to extend a derivation, a tableau calculus usually looks at the literals found on a branch. This is not true for the Link rule of our calculus, which works on literals in origin clauses instead of literals on the branch. However, we would still like to find possible applications of Link by looking at the branch literals alone. The lemma below, used in the proof of Lemma 6.10, allows us to do exactly that.

Lemma 6.5 *A Link rule application with clause $A \to B$ is possible on a branch b if there exist*

1. *literals $L_1, \ldots, L_n \in \ell(b)$ and*
2. *a limited universal unifier σ^a with $A\sigma^a = \{L_1\varrho_1, \ldots, L_n\varrho_n\}\sigma^a$ using refreshing substitutions $\varrho_1, \ldots, \varrho_n$*
3. *such that for some i, $L_i\varrho_i\sigma^a \not\sim_f L_i\varrho_i$.* □

(The reverse does not hold.)

Proof *Let the premises of the lemma be given. The Link rule (Def. 4.5) requires that there be literals $L'_1, \ldots, L'_n$ from clauses $C_1, \ldots, C_n$ such that for every i, $OR_b(L_i, C_i)$ and $L'_i \succ L_i$ holds, L_i being the literal from the premises. Also, we need to find a universal substitution τ^a unifying $\{L'_1\varrho_1, \ldots, L'_n\varrho_n\}$ and A. Last, we have to ensure that for some i, $L'_i\varrho_i\tau^a \not\sim_f L'_i\varrho_i$. In other words, there must exist literals that will unify with A in the origin clauses of the literals on the branch, such that at least one of the literals in question is instantiated.*

First, we prove that the literals $L'_1, \ldots, L'_n$ exist. For every i, there exists a tableau clause C'_i with $TC_b(L_i, C'_i)$, and for this clause we have $L_i \in Lits(C'_i)$ by the Ext rule (Def. 4.4). Also, by the same definition, for every i there exists an origin clause C_i with $OR_b(L_i, C_i)$. Since the unifier σ^a in the Ext rule is limited, these clauses are related by $C_i \succ C'_i$, which means that there exists a literal $L'_i \in Lits(C_i)$ with $L'_i \succ L_i$.

Next, we need to prove that there exists a universal unifier for the sets $\{L'_1\varrho_1, \ldots, L'_n\varrho_n\}$ and A. Note that since $L'_i \succ L_i$, we have $L'_i\lambda_i\upsilon_i = L_i$ for some limited universal substitution λ_i and witnessing substitution υ_i for all i. The witnesses introduced by the υ_i are all distinct. Since the refreshers rename variables and make all the literals variable-disjoint, also to the literals in A, there similarly exist limited universal resp. witnessing substitutions $\hat{\lambda}_i$ and $\hat{\upsilon}_i$ with disjoint support such that $L'_i\varrho_i\hat{\lambda}_i\hat{\upsilon}_i = L_i\varrho_i$.

We thus get $A\sigma^a = \{L_1\varrho_1, \ldots, L_n\varrho_n\}\sigma^a = \{L'_1\varrho_1\hat{\lambda}_1\hat{\upsilon}_1, \ldots, L'_n\varrho_n\hat{\lambda}_n\hat{\upsilon}_n\}\sigma^a$. Since the supports are disjoint, we can move the $\hat{\lambda}_i$ and $\hat{\upsilon}_i$ out and obtain

$A\sigma^a = \{L'_1\varrho_1, \ldots, L'_n\varrho_n\}\hat{\lambda}_1\hat{\upsilon}_1 \ldots \hat{\lambda}_n\hat{\upsilon}_n\sigma^a$. *As witnessing substitutions commute with limited universal substitutions, we obtain the equality* $A\sigma^a = \{L'_1\varrho_1, \ldots, L'_n\varrho_n\}\hat{\lambda}_1 \ldots \hat{\lambda}_n\sigma^a\hat{\upsilon}_1 \ldots \hat{\upsilon}_n$. *Finally, due to refreshing, the substitutions* $\hat{\lambda}_i$ *have no effect on* A*, and since the witnessing substitutions are all injective, we can reverse their effect and get*

$$A\hat{\lambda}_1 \ldots \hat{\lambda}_n\sigma^a\hat{\upsilon}_1^{-1} \ldots \hat{\upsilon}_n^{-1} = \{L'_1\varrho_1, \ldots, L'_n\varrho_n\}\hat{\lambda}_1 \ldots \hat{\lambda}_n\sigma^a\hat{\upsilon}_1^{-1} \ldots \hat{\upsilon}_n^{-1}$$

which gives us $\tau^a = \hat{\lambda}_1 \ldots \hat{\lambda}_n\sigma^a\hat{\upsilon}_1^{-1} \ldots \hat{\upsilon}_n^{-1}$ *as universal unifier. This means that there is a most general such unifier, which we shall denote* $\hat{\tau}^a$.

Finally, due to item 3 in the premises we have a literal $L_i \prec L'_i$ *with* $L_i\varrho_i\sigma^a \not\sim_f L_i\varrho_i$*, and we would like to prove that* $L'_i\varrho_i\hat{\tau}^a \not\sim_f L'_i\varrho_i$*. Since* σ^a *has this property (item 3) and* $\hat{\lambda}_i$ *is a substitution to make* L'_i *equal to* L_i *together with a witnessing substitution,* τ^a *also has it. Making the unifier an MGU cannot destroy this type of instantiation, so we conclude that* $L'_i\varrho_i\hat{\tau}^a \not\sim_f L'_i\varrho_i$ *and are done.* ■

Before proceeding further, we need to define a way to construct a model from an open branch (Def. 6.8). To do that, we shall need the definition below.

Definition 6.6 (Herbrand universe) *Given a clause set* S*, the* Herbrand universe *of* S*, written* $\mathcal{H}_U(S)$*, is the set of ground terms present in the clauses of* S *(possibly augmented with a constant). For coherent logic this is simply the set of constants in the clauses of* S*.* □

Example 6.7 *The Herbrand universe of* $S = \{Pax \vee Qbx, Qyc\}$ *is* $H_U(S) = \{a, b, c\}$. □

Given these definitions, we are ready to construct a Herbrand model for an open branch by specifying the ground atoms to satisfy. The model construction we use is adapted from [Bau98], which again is similar to the one in the completeness proof for the disconnection calculus [LS07].

Definition 6.8 (Model construction) *Given an open branch* b*, construct a model* $\mathcal{M}$ *as follows:*

1. $|\mathcal{M}| = \mathcal{H}_U(C^+(b))$.
2. $c^{\mathcal{M}} = c$ *for every constant or witness* c *(all ground terms are interpreted as themselves).*
3. *For every ground atom* L^g*, we have* $\mathcal{M} \models L^g$ *iff*

$L^g = L\gamma^a\upsilon$ for some $L \in C \in C^-(b)$, some universal grounding substitution γ^a for C, and some witnessing substitution υ that satisfy the requirements that

a) there is an $L' \in \ell(b)$ with $OR_b(L', C)$ and $L \succ L' \succ L\gamma^a\upsilon$, and

b) there is no $D \in C^-(b)$ with $C\gamma^a \leq_0 D <_0 C$. □

This definition means that only instances of atoms on the branch may be true in our model. The reverse does not hold due to the requirement that there are no clauses in $C^-(b)$ between C and $C\gamma^a$, since we may for some instances of C choose a different disjunct to satisfy. Consider an example that illustrates this.

Example 6.9 *Consider the clause set below together with a tableau derivation.*

Clause set:

(A) $P\dot{u}x$

(B) $P\dot{u}b$

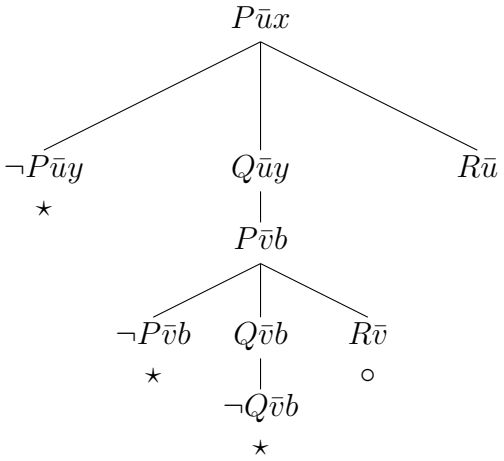
(C) $Pxy \to Qxy \vee Rx$

(D) $Pxb \to Qxb \vee Rx$

(E) $Qxb \to \bot$

(F) $Qay \to \bot$

Derivation:



Note that the derivation is not finished, and the example model we give here does not satisfy the clause set.

Consider the branch marked with ○. This branch is open, so we can construct a model $\mathcal{M}$ from it by Def. 6.8. The domain is $|\mathcal{M}| = \{a, b, \bar{u}, \bar{v}\}$. We have that $\mathcal{M} \models Q\bar{u}a$, because $Q\bar{u}y$ is on the branch with origin clause (C) and literal $Qxy \succ Q\bar{u}y \succ Q\bar{u}a$, and furthermore there is no clause from C^- between the origin clause (C) and $P\bar{u}a \to Q\bar{u}a \vee R\bar{u}$.

Notice that $\mathcal{M}$ does not satisfy $Qa\bar{u}$ even though there is a literal in (C) that can be made equal to it using $\sigma^a = \{x/a, y/\bar{u}\}$, as $Qxy \succ Q\bar{u}y \not\succ Qa\bar{u}$. This is a good thing, for the clause set contains clause $(F) = Qay \to \bot$, and there is no model that can satisfy both $Qa\bar{u}$ and (F).

Now consider $Q\bar{u}b$. We have that $\mathcal{M} \not\models Q\bar{u}b$ (as expected), because, while $Q\bar{u}y$ is on the branch with condition 3a satisfied using $\gamma^a = \{x/\bar{u}, y/b\}$, there is a clause (D) from C^- between (C) and $B\gamma^a = P\bar{u}b \rightarrow Q\bar{u}b \vee R\bar{u}$.

Finally, $\mathcal{M} \models P\bar{u}a$, but not Pab, as there is no universal grounding substitution or witnessing substitution that could make $\dot{u}$ equal to a. □

The idea behind this model definition (and the ones in the works cited above) is that every literal on a branch represents every ground instance of itself. As the example shows, we need to constrain this basic idea to make it work. To do so properly, we can observe that an 'exception' to a literal L from clause C representing some ground instance of itself occurs when this instance is also an instance of some literal that must be false (Pb in the example above). Such a situation can of course be indirect, we could e.g. have had $Pb \rightarrow Rb$ and $Rb \rightarrow \bot$ instead of $Pb \rightarrow \bot$, but this does not matter — there is a path of instances from L to $\bot$.

To pin down the issue, we can notice that if such an exception exists, it involves a clause that leads to a proper instance of L and C. (If the clause lead to C without instantiating it, the branch would have been closed) This means that the Link rule will make sure to add this instance to our clause set, motivating us to look there for such exceptions. Doing this requires some care. First, we need to be certain that such an instance of C actually ends up on the branch we are looking at (or an extension of it), otherwise we would have a model that does not satisfy every clause. This would have happened if we for some reason did not add $Pb \vee Qb$ to the branch, then the model would not have satisfied Pb as before, but it would not have satisfied Qb either. This, however, cannot occur due to the interplay between Link and Ext discussed in Chapter 7.

Second, the presence of existential variables complicates matters. Existential variables become constants during Ext, so if a clause $D \in C^-(b)$ contains existential variables, the tableau clause $D\sigma^a\upsilon$ that we get on the branch is always a proper instance of D — which means that if we talk about tableau clauses, we may well get a model that doesn't satisfy anything at all. The solution is, once again, to talk about the origin clauses, as they are equal in their existential variables. This is the reason behind the 'zero-relation' $\geq_0$, as we do not want Px to be strictly more general than $P\dot{u}$, nor $P\dot{u}$ to make anything except $P\bar{u}$ true.

We connect the above definition of a branch model to tableaux and derivations in the lemma below. The argument is similar to the one in the completeness proof for the NG calculus [Bau98], but we have to deal with two types of variables and a significantly different model construction.

Lemma 6.10 (Model existence) *Let P be a fair tableau derivation for a clause set S. If P is not a refutation, then S is satisfiable.* □

Proof *Let the set S and fair derivation P be given, and assume that P contains a tableau with finished open branch b. Construct a model $\mathcal{M}$ from b as defined in Def. 6.8. We must prove that $\mathcal{M}$ satisfies every clause in S, which we will restate as follows: For every clause $C \in S$ and every universal grounding substitution γ^a for C there exists an existential grounding substitution γ^e for C such that $\mathcal{M}$ satisfies $C\gamma^a\gamma^e$.*

Assume for contradiction that this is not the case, i.e. there exists a clause $C = A \to B \in C^-(b) \supseteq S$ and universal grounding substitution γ^a for C such that $\mathcal{M}$ does not satisfy $C\gamma^a\gamma^e$ for any existential grounding substitution γ^e for C. Pick C to be such that there is no $D \in C^-(b)$ with $C\gamma^a \leq_0 D <_0 C$. We say, borrowing from [Bau98], that C is a most specific generalization of $C\gamma^a$ in $C^-(b)$. This always exists: Any decreasing chain $C >_0 C_1 >_0 C_2 >_0 \cdots$ would eventually reach a term depth exceeding that of $C\gamma^a$.

Using the semantics, the assumption that there is no γ^e such that $\mathcal{M} \models C\gamma^a\gamma^e$ means that $\mathcal{M} \models K^g$ for every ground atom $K^g \in A\gamma^a$, as A contains no existential variables and is therefore grounded by γ^a. Since $\mathcal{M} \models A\gamma^a$, we have from Def. 6.8 that for every $K^g \in A\gamma^a$ there exists an atom $L \in E \in C^-(b)$ and an atom $L' \in \ell(b)$ with $OR_b(L', E)$ such that, for some universal grounding substitution δ^a and witnessing substitution υ it holds that $L \succ L' \succ L\delta^a\upsilon = K^g$. This means that for some limited universal substitution λ^a and witnessing substitution $\hat{\upsilon}$, $L'\lambda^a\hat{\upsilon} = L\delta^a\upsilon = K^g$. Since $L' \in \ell(b)$ contains no existential variables and λ^a is limited, $L\lambda^a\hat{\upsilon} = L\lambda^a$, so we get $L'\lambda^a = K^g$. Doing this for every $K^g \in A\gamma^a$, we get substitutions $\lambda^a_1, \ldots, \lambda^a_n$ such that, $\{L'_1\lambda^a_1, \ldots, L'_n\lambda^a_n\} = A\gamma^a$.

If we apply refreshers ϱ_i to the literals L_i to make them variable-disjoint, pairwise and also with A, we obtain new limited universal substitutions $\hat{\lambda}^a_i$ with disjoint support such that $\{L'_1\varrho_1\hat{\lambda}^a_1, \ldots, L'_n\varrho_n\hat{\lambda}^a_n\} = A\gamma^a$. Due to the variable-disjointness, this implies that $\{L'_1\varrho_1, \ldots, L'_n\varrho_n\}\hat{\lambda}^a_1 \ldots \hat{\lambda}^a_n = A\gamma^a$, and that therefore $\hat{\tau}^a = \hat{\lambda}^a_1 \ldots \hat{\lambda}^a_n\gamma^a$ is a universal unifier for $\{L'_1\varrho_1, \ldots, L'_n\varrho_n\}$ and A. Finally, since there exists a universal unifier for these two sets, there also exists a most general such unifier for them. Call this unifier τ^a.

Since L'_i are literals from the branch and A contains the negative literals of a clause, neither can contain existential variables. Therefore, the MGU τ^a does not introduce existential variables, which means that it is limited.

From here, we need to consider two cases. If τ^a is such that for every i, it holds that $L'_i\varrho_i\tau^a \sim_f L'_i\varrho_i$, an Ext rule application with clause C is possible. If this condition does not hold, by Lemma 6.5 a Link rule application with

clause C is possible, but using a different unifier τ'^a for $\{L_1\varrho_1, \ldots, L_n\varrho_n\}$ and A (cf. Lemma 6.5).

Ext *Since the branch is finished, this application of the Ext rule has happened, and there is an atom from the positive literals of $C\tau^a\upsilon'$ (Def. 4.4) on b. (The positive part of C cannot be empty, for then b would have been closed.) Since $C\tau^a\upsilon' \succ C\gamma^a\upsilon'$, there is now an atom more general than some disjunct of $B\gamma^a\upsilon'$ on the branch, and since there is no existential grounding substitution γ^e such that $\mathcal{M} \models B\gamma^a\gamma^e$, there must be a clause $D \in C^-(b)$ such that $C\gamma^a \leq_0 D <_0 C$, as per Def. 6.8 (C is the origin clause of $C\tau^a\upsilon'$). However, this contradicts our choice of C as the most specific generalization of $C\gamma^a$ in $C^-(b)$.*

Link *Since the branch is finished, this application of the Link rule has happened, and there is for some i a clause $E_i\varrho_i\tau'^a \in C^-(b)$ with $E_i\varrho_i\tau'^a \not\sim_f E_i\varrho_i$, and E_i contains literal L_i that made $\mathcal{M}$ satisfy some $K^g = L_i\delta_i^a\upsilon_i$. As τ'^a is a universal MGU (cf. Lemma 6.5) for $\{L_1\varrho_1, \ldots, L_n\varrho_n\}$ and A, and A contains no existential variables, τ'^a is limited. Furthermore, since $K^g \in A$ is ground we get $E_i\delta_i^a \leq_0 E_i\varrho_i\tau'^a <_0 E_i$ However, this contradicts our assumption that $\mathcal{M} \models L_i\delta_i^a\upsilon_i$, as $E_i\varrho_i\tau'^a$ violates condition 3b in Def. 6.8.*

As both cases end up in a contradiction, we are done. ■

The previous lemma is almost a proof of completeness by itself. However, we tie everything together and prove completeness in the theorem below.

Theorem 6.11 (Completeness) *The presented calculus is complete: If a clause set S is unsatisfiable, then there exists a refutation of S.* □

Proof *Let the satisfiable clause set S be given, and construct a fair tableau derivation P for S using definitions 6.2 and 6.3. Since S is unsatisfiable, by Lemma 6.10 P must be a refutation for S.* ■

6.1 Completeness notes

After completing our proof, we received an e-mail from Peter Baumgartner telling us about a paper by Feng, Sun, and Wu [FSW06] that claims to have found a counterexample to Baumgartner's NG calculus, which our own work is based upon. The counterexample is problem 'MSC006-1' from TPTP [SS98], and according to the paper it admits no refutation in NG. This was a serious issue, as our calculus does not significantly differ from NG for clauses without existential variables. The problem is the following clause set, with a partial derivation.

Clause set:

(A) $Pxy \wedge Pyz \rightarrow Pxz$

(B) $Qxy \wedge Qyz \rightarrow Qxz$

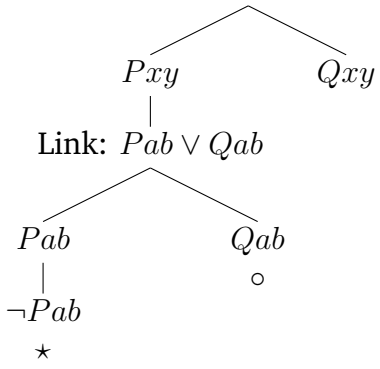
(C) $Qxy \rightarrow Qyx$

(D) $Pxy \vee Qxy$

(E) $Pab \rightarrow \bot$

(F) $Qcd \rightarrow \bot$

Derivation:



We put clause (D) on the branch, then apply Link with Pxy and $Pab \rightarrow \bot$, putting the resulting clause on the branch. Of interest is the branch marked with $\circ$. To get a refutation, we need to expand this branch with clause (A). If we try this, we need to take two copies of Pxy on the branch, renaming the variables, and unify this with $\{Pxy, Pyz\}$ of clause (A). Call the copies $\{Puv, Pws\}$, and compute a unifier $\{x/u, y/v, w/v, x/s\}$. This unifier changes Pws to Pvs, but as Pvs is a renaming of Pws (and likewise for their respective clauses) this does not violate the condition in the Ext rule, neither in NG nor in our calculus. The clause to expand the branch is thus $Puv \wedge Pvs \rightarrow Pus$.

The authors of [FSW06] claim that the inference is redundant, as 'the head literal to extend the branch [Pus]... [is] a variant of the literal Pxy on the branch'. This immediately means that our calculus is not affected, as we have no such definition of a redundant inference. However, on a finer reading of Baumgartner's paper on the NG calculus [Bau98], we discovered that the condition of redundancy he employs demands that the clause to extend the branch is a variant of a *clause* on the branch. It is not sufficient to have a literal be a variant of a literal on the branch. This means that the NG calculus is also complete as presented in [Bau98]. However, for both for our calculus and for NG this demonstrates that a redundancy condition must compare clauses (and not merely literals) to avoid loss of completeness.

Chapter 7

Odds and ends

During our work on this calculus, we noticed a curious effect, referred to in Section 4.1. We call this Link tree, as the Link applications form a tree. This effect is also present in the NG calculus [Bau98], but without direct mention. The effect in question is that a Link rule application always leads to an Ext rule application, sometimes via many intermediary Link rule applications. Consider an illustrative example (refreshers omitted), which works the same way in NG:

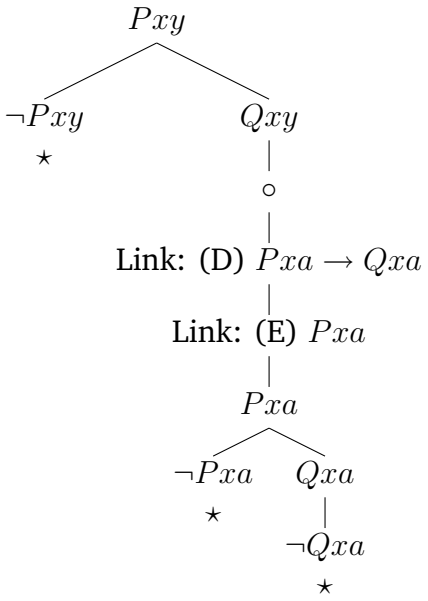
Clause set:

(A) Pxy

(B) $Pxy \to Qxy$

(C) $Qxa \to \bot$

Derivation:



We expand with clauses (A) and (B), then wish to close the right branch by expansion with (C) (at the point marked with $\circ$), which is not possible directly. Instead, we use Link once to generate (D), which does

not help us — however, (D) can be used to Link again, generating (E), which *can* be used to expand the branch. Then we can expand with (E), and finally, with (C). Thus, the Link rule application with (C) leads to an Ext rule application with (C).

To formalize this effect and prove that it is indeed universal, we shall need the following definition.

Definition 7.1 (Branch extension) *Given branches $b = \langle p, N, TC, OR\rangle$ and $b' = \langle p', N', TC', OR'\rangle$, we say that b' is an extension of b if and only if the following conditions hold:*

1. *p is a sublist of p',*
2. *$N \subseteq N'$,*
3. *$TC \subseteq TC'$, and*
4. *$OR \subseteq OR'$.*

In other words, b' extends b if and only if b is a part of b'. Notice that according to this definition, both Link and Ext generate extensions of the branch they are applied on. □

Now, we are ready to formulate and prove a lemma that properly describes the Link tree effect. A similar lemma (if not this one, as is) can be formulated for the NG calculus.

Lemma 7.2 (Link tree) *If a Link rule application with a clause $C = A \to B$ is possible on a branch b as described by Lemma 6.5, there exists an extension of b on which an Ext rule application with C is possible.* □

Proof *Let the branch and the clause be given. From Lemma 6.5 we know that there exist literals $L_1, \ldots, L_n$ on b such that $A\sigma = \{L_1\varrho_1, \ldots, L_n\varrho_n\}\sigma$ using some refreshers $\varrho_1, \ldots, \varrho_n$.*

An Ext rule application with C is possible if there are literals $L'_1, \ldots, L'_n$ on b such that $\{L'_1\varrho'_1, \ldots, L'_n\varrho'_n\}$ and A unify using some refreshers $\varrho'_1, \ldots, \varrho'_n$ without instantiating any of the literals L'_i. We need to find these literals. We shall do so by repeatedly applying the Link rule up the tableau that b is part of until we reach some clauses that can be used in Ext rule applications. Then we shall show that the whole tree of clauses generated by Link can be used to extend the branch b, yielding the desired literals.

Apply Link on b using the clause C. The set $C^-(b)$ is now extended with clauses $D_1, \ldots, D_n$ such that for every $i \leq n$, there exists a literal $L \in \ell(b)$ with origin clause D_i^o, and D_i is a proper instance of D_i^o using some universal

substitution $\hat{\tau}$. We want the negative literals of D_i to unify with literals on b. To see that it does, notice that D_i^o is an origin clause — it has been used in an Ext rule application on b, so its negative literals unify with the branch. Now consider any universal variable x from D_i^o such that $x\hat{\tau}$ is not a universal variable. The term $x\hat{\tau}$ must occur in A, since D_i was produced by unifying some literals from D_i^o with some literals from A. Since both A and the negative literals of D_i^o unify with the branch, the negative literals of D_i must also. As this unifier unifies a negative part of a clause with the branch, it is limited. Thus, if this unifier fulfills the conditions in the Ext rule, then we have found our clauses. If not, then by Lemma 6.5 a Link rule application with every such D_i is possible, and we can repeat this paragraph.

This process must terminate: Since the tableau branch is conversely well-founded, there is a first clause (or clauses) that were used with Ext, and these clauses must have an empty negative part (they must have the form $\top \to B'$). Any instance of such a clause can be used in an Ext rule application at any time.

Let us take the sequence of Link applications above and make it into a tree: At the root, we have C. The children of C are the clauses $D_1, \ldots, D_n$ generated by the Link rule application using C. In general, the children of any node are the clauses generated from a Link rule application with the clause at that node; clauses that cannot be used in a Link rule application have no children. As shown above, this tree is finite.

For every level of the tree with clauses $D_1, \ldots, D_n$, we have that the parent clause D^p unifies with some set of literals from the bodies of $D_1, \ldots, D_n$ using the universal substitution from the Link rule application in question. Now we use an inductive argument to show that the whole tree can be used (incrementally) in Ext rule applications. The base case are the leaves of the tree — the clauses in them are either without negative literals or (as shown above) unifiable with the branch. In either case, they can be used in Ext rule applications directly.

Now assume we have for some node with clause D applied Ext for every child of this node on an extension b' of b. Every such application extends b', generating new branches. Every such branch contains at least one positive literal from every clause in the children. Pick the branch that contains the literals used in the Link rule application with clause D, and call it b''. We know that these literals were made equal to literals in the negative part of D, so the negative part of D unifies with b'', and this unifier does not need to alter the literals on the branch — so an Ext rule application with clause D^p is possible.

Since C is at the root of the tree, an Ext rule application with C is possible on some extension of b. ∎

This effect can be said to be the 'real' reason for completeness of both our calculus as well as of NG. Speaking in a high-level language, we can say that the calculi are complete because

a) If a clause C is falsified by the model from the current branch b, it is always possible to do 'something', and

b) as stated by Lemma 7.2, if the 'something' is a Link rule application, then

c) an Ext rule application is sooner or later possible, which

d) makes the model of an extension of b satisfy C.

It would have been more fulfilling to prove completeness in the way outlined. However, Lemma 7.2 refers to an extension of a branch, while we need an open finished branch for Lemma 6.10. The problem is that there is no way to know, in advance, *which* extension from Lemma 7.2 will become such a branch, a fact that thwarted all our attempts at the kind of proof outlined above. This made us go for the style of proof found in [Bau98], which works just fine.

Chapter 8

Summary and future work

> Ideas won't keep; something must be done about them.
>
> Alfred North Whitehead

In this thesis we have presented an instance-based hyper-tableau calculus for coherent logic based on the next-generation hyper-tableau calculus of Baumgartner [Bau98]. We have described this new calculus in detail, explaining why it is not as easy to define as one might think. Particular care was taken to define it in such a way that neither bookkeeping of dependencies nor complex terms are required at any point. We have also proven it sound and complete. Additionally, we have presented an overview of related methods, both instance-based and otherwise, and discussed their similarities and differences to each other as well as to our own calculus.

The most important direction for future theoretical work will be to consider refinements like the following.

- Only apply Ext with some clause C if the model generated from a branch does not yet satisfy C. To implement this refinement we would need a way to efficiently describe the model and find a way to check whether it satisfies a clause.

- Built-in equality handling. As we have no complex terms, this should be possible in a very efficient manner, as the only possible equalities are $x = y$, $x = a$ and $a = b$ — equalities between variables and constants.

- Efficient model generation from an open branch, e.g. like in the Disconnection calculus [LS07]. The big questions here are about what we can and cannot do with a model description. For example, is it possible to decide validity of ground literals in such a model, or of clauses?

Another obvious direction of work will be to implement the calculus to see if it performs as expected, and what we do and do not gain from the lack of complex terms.

Bibliography

[Ale95] Geoffrey D. Alexander. *Proving first-order equality theorems with hyper-linking*. PhD thesis, The University of North Carolina at Chapel Hill, 1995.

[Ale97] Geoffrey D. Alexander. CLIN-E — Smallest Instance First Hyper-Linking. *Journal of Automated Reasoning*, 18(2):177–182, 1997.

[AP92] Geoffrey D. Alexander and David A. Plaisted. Proving equality theorems with hyper-linking. In *CADE-11: Proceedings of the 11th International Conference on Automated Deduction*, pages 706–710, London, UK, 1992. Springer-Verlag.

[Bau98] Peter Baumgartner. Hyper tableau — the next generation. In Harrie de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1397 of *LNCS*, pages 60–76. Springer, 1998.

[Bau00] Peter Baumgartner. FDPLL: A First-Order Davis-Putnam-Logeman-Loveland Procedure. In *Automated Deduction — CADE-17*, volume 1831 of *LNCS*, pages 200–219. Springer, 2000.

[BC05] Marc Bezem and Thierry Coquand. Automating coherent logic. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 of *LNCS*, pages 246–260. Springer, 2005.

[Bez05] Mark Bezem. On the undecidability of coherent logic. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *LNCS*, pages 6–13. Springer, 2005.

[BFdNT09] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 7(1):58–74, 2009. Special Issue: Empirically Successful Computerized Reasoning.

[BFN96] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper tableaux. In *Logics in Artificial Intelligence*, volume 1126 of *LNCS*, pages 1–17. Springer, 1996.

[BFT04] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Darwin: A theorem prover for the model evolution calculus. In Stephan Schulz, Geoff Sutcliffe, and Tanel Tammet, editors, *IJCAR Workshop on Empirically Successful First Order Reasoning (ESFOR (aka S4))*, Electronic Notes in Theoretical Computer Science, 2004.

[BFT06] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Lemma learning in the model evolution calculus. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4246 of *LNCS*, pages 572–586. Springer, 2006.

[Bil96] Jean-Paul Billon. The disconnection method - a confluent integration of unification in the analytic framework. In *TABLEAUX '96: Proceedings of the 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 110–126, London, UK, 1996. Springer-Verlag.

[Bro78] Frank M. Brown. Towards the automation of set theory and its logic. *Artificial Intelligence*, 10(3):281–316, 1978.

[BS] Peter Baumgartner and Gernot Stenz. IJCAR 2004 tutorial T3: Instance based methods. Currently availible at http://www.mpi-inf.mpg.de/~baumgart/ijcar-tutorial/.

[BT03] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In *Automated Deduction - CADE-19*, volume 2741 of *LNCS*, pages 350–364. Springer, 2003.

[CP94] Heng Chu and David A. Plaisted. Semantically guided first-order theorem proving using hyper-linking. In *Automated Deduction — CADE-12*, volume 814 of *LNCS*, pages 192–206. Springer, 1994.

[CP97] Heng Chu and David A. Plaisted. CLIN-S — A Semantically Guided First-Order Theorem Prover. *Journal of Automated Reasoning*, 18(2):183–188, 1997.

[dNM06] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In John Harrison, Ulrich Furbach, and Natarajan Shankar, editors, *International Joint Conference on Automated Reasoning 2006*, volume 4130 of *LNCS*, page 15 pages. Springer, 2006.

[FH91] H. Fujita and R. Hasegawa. A model generation theorem prover in KL1 using a ramified-stack algorithm. In *Proceedings 8th International Conference on Logic Programming, Paris/France*, pages 535–548. MIT Press, 1991.

[FSW06] Shasha Feng, Jigui Sun, and Xia Wu. Hyper Tableaux — The Third Version. In *Knowledge Science, Engineering and Management*, volume 4092 of *LNCS*, pages 127–138. Springer, 2006.

[Fuc04] Alexander Fuchs. Darwin: A Theorem Prover for the Model Evolution Calculus. Master's thesis, University of Koblenz-Landau, 2004.

[GK03] H. Ganzinger and K. Korovin. New directions in instantiation-based theorem proving. In *Proc. 18th IEEE Symposium on Logic in Computer Science,(LICS'03)*, pages 55–64. IEEE Computer Society Press, 2003.

[GK04] H. Ganzinger and K. Korovin. Integrating equational reasoning into instantiation-based theorem proving. In *Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2004.

[Han04] Christian Mahesh Hansen. Incremental proof search in the splitting calculus. Master's thesis, Dept. of Informatics, University of Oslo, 2004.

[Häh01] Reiner Hähnle. Tableaux and related methods. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 101–178. Elsevier Science B.V., 2001.

[Kor] Konstantin Korovin. An invitation to instantiation-based reasoning: From theory to practice. Currently availible at http://www.cs.man.ac.uk/~korovink/my_pub/index.html.

[Kor08] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning, (IJCAR 2008)*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

[Lee90] Shie-Jue Lee. *CLIN: An automated reasoning system using clause linking*. PhD thesis, The University of North Carolina at Chapel Hill, 1990.

[LP92] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9:25–42, 1992.

[LS01] Reinhold Letz and Gernot Stenz. DCTP — a disconnection calculus theorem prover - system abstract. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 381–385, London, UK, 2001. Springer-Verlag.

[LS02] Reinhold Letz and Gernot Stenz. Integration of equality reasoning into the disconnection calculus. In *TABLEAUX '02: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 176–190, London, UK, 2002. Springer-Verlag.

[LS07] Reinhold Letz and Gernot Stenz. The disconnection tableau calculus. *Journal of Automated Reasoning*, 38:79–126, 2007.

[MB88] Rainer Manthley and François Bry. SATCHMO: A theorem prover implemented in Prolog. In *9th International Conference on Automated Deduction*, volume 310 of *LNCS*, pages 415–434. Springer, 1988.

[Nor07] Jon Malm Norgaard. An automated translation of first-order logic formulas to coherent theories. Master's thesis, University of Bergen, 2007.

[PZ00] David A. Plaisted and Yunshan Zhu. Ordered semantic hyper-linking. *Journal of Automated Reasoning*, 25(3):167–217, 2000.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS06] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[YP02] Adnan Yahia and David A. Plaisted. Ordered semantic hyper tableaux. *Journal of Automated Reasoning*, 29:17–57, 2002.