EPiC Series in Computer Science

Volume 33, 2015, Pages 71–86

ARQNL 2014. Automated Reasoning in Quantified Non-Classical Logics

Theorem Proving for Logic with Partial Functions by Reduction to Kleene Logic [*]

Hans de Nivelle

Instytut Informatyki Uniwersytetu Wrocławskiego
Ul. Joliot-Curie 15, 50-383, Wrocław, Poland
nivelle@ii.uni.wroc.pl

Abstract

We develop a theorem proving strategy for Partial Classical Logic (PCL) that is based on geometric logic. The strategy first translates PCL theories into sets of Kleene formulas. After that, the Kleene formulas are translated into 3-valued geometric logic. The resulting formulas can be refuted by an adaptation of geometric resolution.

1 Introduction

Partial Classical Logic (PCL) was introduced by the author in [6] with the aim of being able to express rich type systems and partial functions. In order to allow type systems of arbitrary complexity, one can use *relativization*. This means that **(1)** ordinary formulas are used for expressing type conditions: $+\colon N \times N \rightarrow N$ can be expressed by $\forall n_1, n_2 \; N(n_1) \wedge N(n_2) \rightarrow N(n_1 + n_2)$. **(2)** Universal quantifiers use implication to include type conditions: $\forall n\colon N \; n \geq 0$ becomes $\forall n \; N(n) \rightarrow n \geq 0$, and **(3)** existential quantifiers use conjunction to include type conditions: $\exists n\colon N \; n \geq 0$ becomes $\exists n \; N(n) \wedge n \geq 0$. The advantage of relativization is the fact that it is flexible: Every property that can be expressed in logic can be used as type condition. The disadvantage is the fact that the type system looses its possibility to pre-check formulas, and to discard meaningless formulas. Using relativization, an ill-typed formula will become either true or false. As a theorem, the formula becomes either too hard to too easy to prove. As an assumption, it will be either too strong or too weak. The dangerous cases, which are an easy theorem, and a strong assumption, may go undetected for long time.

In order to model undefinedness, it has been proposed to use three-valued Kleene logic. In Kleene logic, the set of Booleans is extended with a third truth-value $\mathbf{u}$, which denotes *unknown*. Ill-typed formulas are treated as underspecified. The truth values are assumed to be ordered by $\mathbf{f} < \mathbf{u} < \mathbf{t}$. Disjunction $\oplus$ is defined by picking the rightmost truth-value from its arguments. Conjunction $\otimes$ picks the leftmost truth-value from its arguments. One has $\mathbf{u} \oplus \mathbf{t} = \mathbf{t}$, which reflects the fact that $\mathbf{u}$ can be interpreted as underspecified. If one of the arguments of a

[*]This work was supported by the Polish National Science Center (Narodowe Centrum Nauki) through grant number DEC-2011/03/B/ST6/00346.

disjunction is $\mathbf{t}$, then the other argument does not matter. Kleene logic has been proposed for modeling partial functions in [9, 11, 10, 13] and [5].

In our view, Kleene logic does not capture the way in which types are intuitively used: Kleene logic is designed in such a way that one can derive as much as possible from underspecified formulas. The fact that $\oplus$ ignores $\mathbf{u}$ when the other argument is $\mathbf{t}$, (the same happens when one of the arguments of $\otimes$ is $\mathbf{f}$) reflects this. As a consequence, it is possible to assume an ill-typed formula. The formula will be assumed true, and it will be possible to derive consequences from it. If the formula is ill-typed, its type correctness is assumed together with the truth of the formula, so that an assumed formula may implicitly declare symbols.

PCL in contrast has *type strictness*: Nothing can be done with a formula when it is ill-typed at the point where it is used. In order to ensure this, contexts in PCL are ordered, and implication and conjunctions are both separated into two operators: the strict operators $\rightarrow$ and $\wedge$, and the lazy operators $[\,]$ and $\langle\,\rangle$. The strict operators are used in formulas in the usual way. A strict operator is well-typed only if both of its arguments are well-typed. The lazy operators are used in relativizations. A lazy operator is well-typed if its left argument is false, or its left argument is true, and its right argument is well-typed.

The goal of the current paper is to sketch a theorem proving procedure for PCL. The procedure is based on geometric logic. The current paper is a shortened version of [7], which contains a more detailed discussion of PCL, and all the proofs. We will define the syntax and semantics of PCL, give examples, and explain how PCL contexts can be decomposed into sets of sequents. (which can be alternatively viewed as sets of formulas.) In Section 2, we introduce the notion of widening, which is the key to obtaining theorem proving procedures for PCL. Widening is also important for understanding the relation between PCL, Kleene logic and classical logic, but we will not discuss this in this paper. It is discussed in [7]. In Section 3, we sketch a theorem proving procedure, based on [8], for three-valued, Kleene logic. In Section 4, we explain the last step of the transformation from sequents into geometric logic. We now define the syntax, and the semantics of PCL:

Definition 1.1. *We first define* terms*: If f is a function symbol with arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term as well. Using terms, we define the set of formulas as follows:*

- *$\bot, E$ and $\top$ are formulas.*
- *If t_1 and t_2 are terms, then $t_1 \approx t_2$ is a formula.*
- *If p is a predicate symbol with arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula.*
- *If F is a formula, then $\neg F$ and $\# F$ are formulas.*
- *If F and G are formulas, then $F \vee G$, $F \wedge G$, $F \rightarrow G$, and $F \leftrightarrow G$ are formulas.*
- *If F and G are formulas, then $\langle F \rangle G$ and $[F]G$ are formulas.*
- *If F is a formula, and x is a variable, then $\forall x\ F$ and $\exists x\ F$ are formulas.*

Definition 1.2. *An* interpretation *is an object of form $I = (D, [\,])$. The set D is the domain of the interpretation. The function $[\,]$ interprets function symbols as functions from D^n to D and predicate symbols as functions from D^n to $\{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$ in accordance with the arity of the symbol.*

Definition 1.3. *Let $I = (D, [\,])$ be an interpretation. Starting with $[\,]$, we define the interpretation function $I()$, that interprets all terms and formulas.*

- *If f is a function symbol of arity n, and $t_1, \ldots, t_n$ are terms, then $I(\ f(t_1, \ldots, t_n)\ ) = [f](I(t_1), \ldots, I(t_n))$.*

- *If p is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms, then* $I(\ p(t_1, \ldots, t_n)\ ) = [p](I(t_1), \ldots, I(t_n))$.
- $I(\bot) = \mathbf{f}$, $I(E) = \mathbf{e}$, *and* $I(\top) = \mathbf{t}$.
- *For a unary propositional operator $\star$, the interpretation of $I(\star A)$ is defined from the value in the corresponding table below, using the value of $I(A)$ in the order* $\mathbf{f}, \mathbf{e}, \mathbf{t}$:

$\neg$:

t	**e**	**f**

$\#$:

t	**f**	**t**

- *For a binary propositional operator $\star$, the interpretation of $I(A \star B)$ is defined from the value in the corresponding table below. The row is determined by the value of $I(A)$ in the order* $\mathbf{f}, \mathbf{e}, \mathbf{t}$, *and the column is determined by the value of $I(B)$, also in the order* $\mathbf{f}, \mathbf{e}, \mathbf{t}$.

$\vee$:

f	**e**	**t**
e	**e**	**e**
t	**e**	**t**

$\wedge$:

f	**e**	**f**
e	**e**	**e**
f	**e**	**t**

$\leftrightarrow$:

t	**e**	**f**
e	**e**	**e**
f	**e**	**t**

$\rightarrow$:

t	**e**	**t**
e	**e**	**e**
f	**e**	**t**

$\langle\ \rangle$:

f	**f**	**f**
e	**e**	**e**
f	**e**	**t**

$[\ ]$:

t	**t**	**t**
e	**e**	**e**
f	**e**	**t**

- *For a quantifier Q, the value of $I(Qx\ F)$ is obtained as follows: First define $R_F = \{ I_d^x(F) \mid d \in D \}$. Then $I(Qx\ F)$ is obtained by selecting from R_F the most preferred value using the preference list for Q :*

$$\forall : \mathbf{e} > \mathbf{f} > \mathbf{t}, \qquad \exists : \mathbf{e} > \mathbf{t} > \mathbf{f}.$$

Theories are constructed in *contexts*. A context is a mixture of assumptions and theorems. Assumptions are assumptions in the usual sense, and type specifications. The order of formulas in a context is essential, because theorems must be provable from the formulas that occur before them, and types of functions and predicates have to be specified by assumptions before they are used.

Definition 1.4. *We call an object of form $\|\Gamma_1, \ldots, \Gamma_m\|$, in which Γ_j $(m \geq 0)$ are formulas, and in which some of the Γ_j are possibly marked with a ϑ, a* context.

Formulas that are marked with ϑ denote theorems, which means that they must be provable from the formulas occurring before them. Unmarked formulas are assumptions.

Example 1.5. $\|\ \#A,\ \#B,\ A,\ B,\ (A \wedge B)^\vartheta\ \|$ *is an example of a context. The formula $(A \wedge B)^\vartheta$ is marked as theorem, which is correct, because it is provable from the formulas A and B. The formulas A and B can be assumed because $\#A$ and $\#B$ occur before them.*

Definition 1.6. *Let $\|\Gamma\|$ be a context. We say that $\|\Gamma\|$ is* strongly valid *if in every interpretation I, for which there is an i, s.t. $I(\Gamma_i) \neq \mathbf{t}$, the first such i satisfies the following condition:*

- *Γ_i is not marked as theorem and $I(\Gamma_i) = \mathbf{f}$.*

Strong validity captures the intuition that theorems must be provable from the formulas before them, and assumptions must be well-defined assuming the formulas before them.

Example 1.7. *The context* $\| \ A, \ B, \ (A \wedge B)^{\vartheta} \ \|$ *is not strongly valid, because Definition 1.6 is not met by* I *with* $I(A) = \mathbf{e}, \ I(B) = \mathbf{t}$. *The context* $\| \ \#A, \ A, \ B, \ (A \wedge B)^{\vartheta} \ \|$ *is still not strongly valid, because it is possible that* $I(B) = \mathbf{e}$, *which would make* B *with* $I(B) \neq \mathbf{f}$ *the first formula with* $I(B) \neq \mathbf{t}$.

In order to obtain a context that is strongly valid, we also have to add $\#B$, *so that we get* $\| \ \#A, \ \#B, \ A, \ B, \ (A \wedge B)^{\vartheta} \ \|$.

Example 1.8. *The context* $\| \ G(s), \ \forall x \ G(x) \rightarrow M(x), \ M^{\vartheta}(s) \ \|$ *is not strongly valid, despite the fact that* $G(s), \ \forall x \ G(x) \rightarrow M(x)$ *implies* $M(s)$.

In order to make the context strongly valid, one has to make sure that predicates G *and* M *(Greek and Mortal) are always well-defined:* $\| \ \forall x \ \#G(x), \ \forall x \ \#M(x), \ G(s), \ \forall x \ G(x) \rightarrow M(x), \ M^{\vartheta}(s) \ \|$. *The predicates* G *and* M *are currently too general. In order to be more realistic, they can be restricted to be subpredicates of a predicate* H *(Human):*

$$\left\| \begin{array}{lll} \forall x \ \#H(x), & \forall x \ H(x) \rightarrow \#G(x), & \forall x \ H(x) \rightarrow \#M(x) \\ \langle H(s) \rangle \ G(s), & \forall x \ [H(x)] \ G(x) \rightarrow M(x), & M^{\vartheta}(s) \end{array} \right\|$$

If the formula $\langle H(s) \rangle \ G(s)$ *would be replaced by* $H(s) \wedge G(s)$, *the resulting context would be not strongly valid anymore, because one can have* $I(H(s)) = \mathbf{f}, \ I(G(s)) = \mathbf{e}$, *while making all the formulas before it true.*

Similarly, replacing $\forall x \ [H(x)] \ G(x) \rightarrow M(x)$ *by* $\forall x \ H(x) \wedge G(x) \rightarrow M(x)$ *would make the context not strongly valid anymore, because there could exist a term* t *with* $I(H(t)) = \mathbf{f}, \ I(G(t)) = \mathbf{e}$.

We now show that a context can be decomposed into a set of sequents. An assumption results in one sequent, representing type correctness of the assumed formula. A theorem results in two sequents, representing type correctness and correctness of the theorem.

Definition 1.9. *A* sequent *is an object of form* $S \vdash \bot$, *in which* S *is a set of formulas.*

Definition 1.10. *Let* $\|\Gamma\| = \|\Gamma_1, \ldots, \Gamma_m\|$ *be a context. The* sequent expansion $\mathrm{Seq}(\ \|\Gamma\| \ )$ *of* $\|\Gamma\|$ *is defined as* $\bigcup_{1 \leq i \leq m} E(\ \|\Gamma\|, \ i) \cup \bigcup_{1 \leq i \leq m} E_{\vartheta}(\ \|\Gamma\|, \ i)$, *where*

- $E(\ \|\Gamma\|, \ i) = \{ \ \{\Gamma_1, \ldots, \Gamma_{i-1}, \neg(\#\Gamma_i)\} \vdash \bot \ \}$.
- $E_{\vartheta}(\ \|\Gamma\|, \ i) = \{ \ \{\Gamma_1, \ldots, \Gamma_{i-1}, \neg\Gamma_i\} \vdash \bot \ \}$ *if* Γ_i *is marked as theorem, and* $\{ \ \}$ *otherwise.*

Definition 1.11. *We say that a set of formulas* S *is* strongly unsatisfiable *if for every interpretation* I, *there is a formula* $A \in S$, *s.t.* $I(A) = \mathbf{f}$.

We say that a set of sequents $S_1 \vdash \bot, \ldots, S_n \vdash \bot$ strongly represents a property P *iff*

1. P *is true implies that all of the* S_i *are strongly unsatisfiable.*
2. P *is false implies that (at least) one of the* S_i *is satisfiable.*

The advantage of the use of strong representation is that it makes it possible to ignore error values during proof search. Either, it is possible to make all formulas in all sequents true, or there always is a false formula in some sequent.

Theorem 1.12. *For every context* Γ, *the sequent expansion* $\mathrm{Seq}(\|\Gamma\|)$ *strongly represents the property that* $\|\Gamma\|$ *is strongly valid.*

For the proof, we refer to [7].

Example 1.13. *Suppose we want to prove that the context of Example 1.8 is strongly valid. First define:*

$$\begin{array}{lllllll} A_0 & = & \forall x\ \#H(x) & & B_0 & = & \neg\#\ \forall x\ \#H(x) \\ A_1 & = & \forall x\ H(x) \rightarrow \#G(x) & & B_1 & = & \neg\#\ \forall x\ H(x) \rightarrow \#G(x) \\ A_2 & = & \forall x\ H(x) \rightarrow \#M(x) & & B_2 & = & \neg\#\ \forall x\ H(x) \rightarrow \#M(x) \\ A_3 & = & \langle H(s)\rangle\ G(s) & & B_3 & = & \neg\#\ \langle H(s)\rangle\ G(s) \\ A_4 & = & \forall x\ [H(x)]\ G(x) \rightarrow M(x) & & B_4 & = & \neg\#\ \forall x\ [H(x)]\ G(x) \rightarrow M(x) \\ G & = & \neg M(s) & & B_5 & = & \neg\#M(s) \end{array}$$

The sequent expansion consists of the following sequents:

$$\begin{array}{ll} B_0 \vdash \bot & A_0, A_1, A_2, A_3, B_4 \vdash \bot \\ A_0, B_1 \vdash \bot & A_0, A_1, A_2, A_3, A_4, B_5 \vdash \bot \\ A_0, A_1, B_2 \vdash \bot & A_0, A_1, A_2, A_3, A_4, G \vdash \bot \\ A_0, A_1, A_2, B_3 \vdash \bot & \end{array}$$

The last two sequents in the second column originate from $M^{\vartheta}(s)$. It created two sequents because it is a theorem.

2 Transformation to Kleene Logic

Decomposition Seq can be used to decompose a context into a set of sequents. By Theorem 1.12, this set of sequents strongly represents strong validity of the original context. In this section, we transform the resulting set of sequents into a set of sequents in Kleene logic, that still strongly represents strong validity the original context.

The notion of strong representation contains a gap between the property being true and the property being false. When the property being represented is true, every sequent always contains a false formula. When the property is false, there is a sequent that can contain only true formulas in some interpretation. This means that strong representation says nothing about **e**, and there is no need to preserve it during transformations. As a consequence, transformations do not need to preserve all truth-values. The following property is sufficient:

Definition 2.1. *We define the* widening relation $\preceq$ *as follows: $A \preceq B$ if in every interpretation I, we have $I(A) = \mathbf{f} \Rightarrow I(B) = \mathbf{f}$, and $I(A) = \mathbf{t} \Rightarrow I(B) = \mathbf{t}$.*

We write $A \equiv B$ if $A \preceq B$ and $B \preceq A$. It can be easily checked that $A \equiv B$ iff $I(A) = I(B)$ in every interpretation I.

Theorem 2.2. *Let $\{S_1 \vdash \bot,\ \dots,\ S_n \vdash \bot\}$ be a set of sequents that strongly represents a property P. Let A and B be two formulas with $A \preceq B$.*

Let $\{S'_1 \vdash \bot,\ \dots,\ S'_n \vdash \bot\}$ be obtained from $\{S_1 \vdash \bot,\ \dots,\ S_n \vdash \bot\}$ by possibly replacing some occurrences of A by B.

Then $\{S'_1 \vdash \bot,\ \dots,\ S'_n \vdash \bot\}$ strongly represents property P as well.

The proof is given in [7]. Since we have now established that one can use $\preceq$ in transformations, there is no need anymore to distinguish between PCL operators and Kleene operators. As a consequence, PCL operators can be replaced by their Kleene counterparts. We will use the notation $\otimes$ for Kleene conjunction, $\oplus$ for Kleene disjunction, Π for Kleene universal quantification, and Σ for Kleene existential quantification.

Definition 2.3. *We extend the set of formulas, defined in Definition 1.1, with two binary operators and two quantifiers as follows: If F and G are formulas, then $F \otimes G$ and $F \oplus G$ are formulas as well. If F is a formula and x is a variable, then $\Pi x\ F$ and $\Sigma x\ F$ are formulas. We extend the interpretation of formulas, defined in Definition 1.3, as follows:*

- *The semantics of the binary operators $\otimes$ and $\oplus$ is defined by the following truth tables:*

$\otimes$:

f	**f**	**f**
f	**e**	**e**
f	**e**	**t**

$\oplus$:

f	**e**	**t**
e	**e**	**t**
t	**t**	**t**

- *The semantics of the quantifiers Π and Σ is defined by the following preferences:*

$$\Pi : \mathbf{f} > \mathbf{e} > \mathbf{t}, \qquad \Sigma : \mathbf{t} > \mathbf{e} > \mathbf{f}.$$

The first row of Figure 1 lists the rules that replace PCL operators by their corresponding Kleene operators. The rules in the second row can be used for pushing the #-operator inward. Operator # is intended to be used for declaring symbols, as in $\forall mn\ N(m) \wedge N(n) \rightarrow \#(m \leq n)$. Because of this, we expect application of # on compound formulas to be rare, so that the rules in the second column probably will not be used often. We included them for completeness.

Theorem 2.4 states that the replacements of Figure 1 agree with $\preceq$ when they are made on top level. Theorem 2.5 states that $\preceq$ can be lifted into formula contexts that do not contain #, so that replacements can be made on subformulas as well.

Theorem 2.4. *For every rule of Figure 1, that is written as $A \preceq B$, it is indeed the case that $A \preceq B$. For every rule that is written as $A \equiv B$, it is indeed the case that $A \equiv B$.*

The propositional cases can be checked by case analysis.

Theorem 2.5. *All of the logical operators of PCL and the Kleene operators, with the exception of #, are monotone relative to $\preceq$. More precisely:*

- *If $A \preceq B$, then $\neg A \preceq \neg B$.*
- *If $A_1 \preceq B_1$ and $A_2 \preceq B_2$, then*

$$\begin{array}{lll} A_1 \wedge A_2 \preceq B_1 \wedge B_2, & A_1 \vee A_2 \preceq B_1 \vee B_2, & A_1 \rightarrow A_2 \preceq B_1 \rightarrow B_2, \\ A_1 \leftrightarrow A_2 \preceq B_1 \leftrightarrow B_2, & [A_1]A_2 \preceq [B_1]B_2, & \langle A_1 \rangle A_2 \preceq \langle B_1 \rangle B_2. \end{array}$$

- *If $P[x] \preceq Q[x]$, then*

$$\begin{array}{ll} \forall x\ P[x] \preceq \forall x\ Q[x], & \exists x\ P[x] \preceq \exists x\ Q[x], \\ \Pi x\ P[x] \preceq \Pi x\ Q[x], & \Sigma x\ P[x] \preceq \Sigma x\ Q[x]. \end{array}$$

It can be easily seen that # is not monotone: For example, one has $E \preceq \top$, but not $\#E \preceq \#\top$.

At this point, we have established that the rules in Figure 1 can be used to rewrite a formula into a normal form. The restriction to contexts not containing # is unproblematic, because # can be pushed inwards down to the atoms.

Definition 2.6. *For a formula A in PCL (possibly mixed with Kleene operators), we define the* Kleening *of A, written as* Kl(A), *as the result of normalizing A using the rules for replacing PCL operators, and the rules for # in Figure 1.*

Figure 1: Kleening Rules

Rules for replacing PCL operators (left) and NNF rules (right):

$A \wedge B$	$\preceq$	$A \otimes B$	$\neg\bot$	$\equiv$	$\top$
$A \vee B$	$\preceq$	$A \oplus B$	$\neg E$	$\equiv$	E
$A \rightarrow B$	$\preceq$	$\neg A \oplus B$	$\neg\top$	$\equiv$	$\bot$
$A \leftrightarrow B$	$\preceq$	$(\neg A \vee B) \otimes (A \vee \neg B)$	$\neg\neg A$	$\equiv$	A
$[A]B$	$\preceq$	$\neg A \oplus B$	$\neg(A \oplus B)$	$\equiv$	$\neg A \otimes \neg B$
$\langle A\rangle B$	$\preceq$	$A \otimes B$	$\neg(A \otimes B)$	$\equiv$	$\neg A \oplus \neg B$
$\forall x\ F[x]$	$\preceq$	$\Pi x\ F[x]$	$\neg(\ \Pi x\ F[x]\ )$	$\equiv$	$\Sigma x\ \neg F[x]$
$\exists x\ F[x]$	$\preceq$	$\Sigma x\ F[x]$	$\neg(\ \Sigma x\ F[x]\ )$	$\equiv$	$\Pi x\ \neg F[x]$

Rules for #:

$\#\top$	$\equiv$	$\top$	$\#(A \wedge B)$	$\equiv$	$\#A\ \otimes\ \#B$
$\#E$	$\equiv$	$\bot$	$\#(A \vee B)$	$\equiv$	$\#A\ \otimes\ \#B$
$\#\bot$	$\equiv$	$\top$	$\#(A \rightarrow B)$	$\equiv$	$\#A\ \otimes\ \#B$
$\#(\neg A)$	$\equiv$	$\#A$	$\#(A \leftrightarrow B)$	$\equiv$	$\#A\ \otimes\ \#B$
$\#(\#A)$	$\equiv$	$\top$	$\#(\ [A]B\ )$	$\equiv$	$\#A\ \otimes\ (\ \neg A \vee \#B\ )$
$\#(\ \forall x\ F[x]\ )$	$\equiv$	$\Pi x\ \#F[x]$	$\#(\ \langle A\rangle B\ )$	$\equiv$	$\#A\ \otimes\ (\ \neg A \vee \#B\ )$
$\#(\ \exists x\ F[x]\ )$	$\equiv$	$\Pi x\ \#F[x]$	$\#(t_1 \approx t_2)$	$\equiv$	$\top$

It is possible to define $\mathrm{Kl}(A)$ by a single, recursive definition. Such definition is given in [7].

Theorem 2.7. *For every formula A, we have $A \preceq \mathrm{Kl}(A)$.*

Theorem 2.7 follows from Theorem 2.4 and Theorem 2.5.

Definition 2.8. *We say that a formula A is* in Kleene logic *if it contains only logical operators from $\bot, \top, E, \neg, \#, \otimes, \oplus, \Pi, \Sigma$ (and $\approx$), and the operator # is applied only on non-equality atoms.*

It is easily checked that $\mathrm{Kl}(A)$ is always in Kleene logic. The rules for $\mathrm{Kl}(\ A \leftrightarrow B\ )$, $\mathrm{Kl}^{\#}(\ \langle A\rangle B\ )$, and $\mathrm{Kl}^{\#}(\ [A]B\ )$ may cause an exponential increase in size of the formula. This problem can be avoided by using suitable subformula replacement rules.

During the rest of the tranformation, it is convenient to push negation inwards as far as possible. This has the advantage that the polarity of non-trivial subformulas is always positive, which will simplify the remaining transformations.

Definition 2.9. *Let A be a Kleene formula. We say that A is* in negation normal form (NNF) *if negation is applied only on atoms and on formulas of form $\#p(t_1, \ldots, t_n)$.*

Definition 2.10. *We define the* negation normal form *of A, for which we write* NNF(A), *as the normal form that is obtained when A is normalized using the NNF rules in Figure 1.*

As with Kl, the negation normal form can be defined in a single recursive definition.

Example 2.11. *We apply Kleening and compute the negation normal form of the formulas in*

Example 1.13.

$$\begin{array}{lcl@{\qquad}lcl}
A_0 & \preceq & \Pi x\ \#H(x) & B_0 & \preceq & \Sigma x\ \bot \\
A_1 & \preceq & \Pi x\ (\neg H(x) \oplus \#G(x)) & B_1 & \preceq & \Sigma x\ (\neg\#H(x) \oplus \bot) \\
A_2 & \preceq & \Pi x\ (\neg H(x) \oplus \#M(x)) & B_2 & \preceq & \Sigma x\ (\neg\#H(x) \oplus \bot) \\
A_3 & \preceq & H(s) \otimes G(s) & B_3 & \preceq & \neg\#H(s) \oplus (H(s) \otimes \neg\#G(s)) \\
A_4 & \preceq & \Pi x\ (\neg H(s) \oplus & B_4 & \preceq & \Sigma x\ (\neg\#H(x) \oplus (H(x) \otimes \\
 & & \quad \neg G(s) \oplus M(s)) & & & \quad (\neg\#G(x) \oplus \neg\#M(x)))) \\
G & \preceq & \neg M(s) & B_5 & \preceq & \neg\#M(s)
\end{array}$$

The Kleening transformation forgets type information. For example, the PCL formulas $\forall x\ [H(x)]\ G(x) \rightarrow M(x)$, $\forall x\ [H(x) \wedge G(x)]\ M(x)$, $\forall x\ H(x) \wedge G(x) \rightarrow M(x)$, $\forall x\ H(x) \rightarrow [G(x)]\ M(x)$, $\forall x\ [H(x)]\ [G(x)]\ M(x)$ all have the same Kleening $\Pi x\ \neg H(x) \oplus \neg G(x) \oplus M(x)$.

The formulas were different in PCL, but Kleening has widened them into the same formula. The formulas F still have different meanings in PCL, because the Kleenings of the formulas $\neg\#F$ differ, so that they will be typechecked in different ways.

The fact that Kleening removes type information can be reformulated as: Once a formula has been type checked, its type information can be forgotten. Alternatively, one can say: A typechecked formula can be considered equivalent to its Kleening. In the full paper [7], it is argued that in most cases, the Kleene translation of a theorem is equivalent to its classical translation, so that one obtains: A typechecked formula can be considered equivalent to its relativization in classical logic. Since people tend to think of their favorite formulas, which are always well-typed, people tend to believe that all type information can be expressed by relativization. In practice, many formulas in applications are not particularly liked by anyone, and for those, type checking will be very useful. The next step in the transformation is called *radicalization*:

Definition 2.12. *For each predicate symbol p with arity n, we define the following abbreviations:*

$$\begin{cases}
p_{\emptyset}(t_1,\ldots,t_n) & := \ \bot \\
p_{\mathbf{f}}(t_1,\ldots,t_n) & := \ \#p(t_1,\ldots,t_n) \otimes \neg p(t_1,\ldots,t_n) \\
p_{\mathbf{e}}(t_1,\ldots,t_n) & := \ \neg\#p(t_1,\ldots,t_n) \\
p_{\mathbf{t}}(t_1,\ldots,t_n) & := \ \#p(t_1,\ldots,t_n) \otimes p(t_1,\ldots,t_n) \\
p_{\mathbf{f},\mathbf{e}}(t_1,\ldots,t_n) & := \ \neg\#p(t_1,\ldots,t_n) \oplus \neg p(t_1,\ldots,t_n) \\
p_{\mathbf{e},\mathbf{t}}(t_1,\ldots,t_n) & := \ \neg\#p(t_1,\ldots,t_n) \oplus p(t_1,\ldots,t_n) \\
p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n) & := \ \#p(t_1,\ldots,t_n) \\
p_{\mathbf{f},\mathbf{e},\mathbf{t}}(t_1,\ldots,t_n) & := \ \top
\end{cases}$$

Theorem 2.13. *For every propositional atom $p_{\lambda}(t_1,\ldots,t_n)$, for every interpretation I, we have $I(p_{\lambda}(t_1,\ldots,t_n)) \in \{\mathbf{f},\mathbf{t}\}$.*

Definition 2.14. *Let A be a Kleene formula in NNF. We recursively define* the radicalization

Rad(A) *of* A *as follows:*

$$\begin{array}{lcl}
\text{Rad}(\ p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{t}}(t_1,\ldots,t_n) \\
\text{Rad}(\neg p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{f}}(t_1,\ldots,t_n) \\
\text{Rad}(\#p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n) \\
\text{Rad}(\#p(t_1,\ldots,t_n)\ ) & = & p_{\mathbf{e}}(t_1,\ldots,t_n) \\
 & & \\
\text{Rad}(\ t_1 \approx t_2\ ) & = & t_1 \approx t_2 \\
 & & \\
\text{Rad}(\top) & = & \top \\
\text{Rad}(E) & = & \bot \\
\text{Rad}(\bot) & = & \bot \\
\text{Rad}(A \otimes B) & = & \text{Rad}(A) \otimes \text{Rad}(B) \\
\text{Rad}(A \oplus B) & = & \text{Rad}(A) \oplus \text{Rad}(B) \\
 & & \\
\text{Rad}(\Pi x\ P[x]) & = & \Pi x\ \text{Rad}(P[x]) \\
\text{Rad}(\Sigma x\ P[x]) & = & \Sigma x\ \text{Rad}(P[x])
\end{array}$$

Radicalization is called 'radicalization' because in the resulting formula, every non-atomic subformula always has a definite truth value. This has a surprising consequence, namely there is no need to use Kleene operators anymore. In the resulting formula, each Kleene operator can be replaced by its corresponding classical (or PCL) operator without changing the truth value of the formula. This implies that in the last four rules of the definition of Rad, one could have used $\wedge, \vee, \forall, \exists$ instead of $\otimes, \oplus, \Pi, \Sigma$.

Theorem 2.15. *For every Kleene formula A in NNF, we have $A \preceq \text{Rad}(A)$.*

Proof. First check (by case analysis) that

$$\begin{array}{lcl}
p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{t}}(t_1,\ldots,t_n), \\
\neg p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{f}}(t_1,\ldots,t_n), \\
\#p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{f},\mathbf{t}}(t_1,\ldots,t_n), \\
\neg\#p(t_1,\ldots,t_n) & \preceq & p_{\mathbf{e}}(t_1,\ldots,t_n), \\
E & \preceq & \bot.
\end{array}$$

After that, apply Theorem 2.5. □

We give an example of radicalization:

Example 2.16. *Radicalizing the formulas in Example 2.11 gives:*

$$\begin{array}{lcl}
A_0 & = & \Pi x\ H_{\mathbf{f},\mathbf{t}}(x) \\
A_1 & = & \Pi x\ (H_{\mathbf{f}}(x) \oplus G_{\mathbf{f},\mathbf{t}}(x)) \\
A_2 & = & \Pi x\ (H_{\mathbf{f}}(x) \oplus M_{\mathbf{f},\mathbf{t}}(x)) \\
A_3 & = & H_{\mathbf{t}}(s) \otimes G_{\mathbf{t}}(s) \\
A_4 & = & \Pi x\ (H_{\mathbf{f}}(s) \oplus G_{\mathbf{f}}(s) \oplus M_{\mathbf{t}}(s)) \\
G & = & M_{\mathbf{f}}(s)
\end{array}
\qquad
\begin{array}{lcl}
B_0 & = & \Sigma x\ \bot \\
B_1 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus \bot) \\
B_2 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus \bot) \\
B_3 & = & H_{\mathbf{e}}(s) \oplus (H_{\mathbf{t}}(s) \otimes G_{\mathbf{e}}(s)) \\
B_4 & = & \Sigma x\ (H_{\mathbf{e}}(x) \oplus (H_{\mathbf{t}}(x) \otimes \\
 & & \quad (G_{\mathbf{e}}(x) \oplus M_{\mathbf{e}}(x)))) \\
B_5 & = & M_{\mathbf{e}}(s)
\end{array}$$

The resulting sequents still strongly represent strong validity of the context in Example 1.8.

At this point, we are close to classical logic, so that it would be possible to define a resolution procedure in the standard way. One can Skolemize the radicalized sequents, factor them into clausal normal form, and apply resolution between atoms $p_\lambda(t_1, \ldots, t_n)$ and $p_\mu(u_1, \ldots, u_n)$, if $\lambda \cap \mu = \emptyset$, and the t_i are unifiable with u_i. Since we are interested in using geometric logic, for reasons that were explained in [8], we will develop a theorem proving procedure for geometric logic in the remaining section of this paper.

3 Three-Valued Geometric Logic

Geometric logic for theorem proving was introduced in [3]. The proof search algorithm for geometric logic is closely related to model generation (see [12]) or to proof search based on hyper tableaux (see [2]). In [8], we introduced a variant of geometric logic, in which function symbols are replaced by predicates, which is able to deal directly with equality, and which learns a lemma, whenever it closes a branch in the search tree. The search algorithm is similar to the algorithm in [4]. Whenever the algorithm encounters an existential quantifier, it first tries all existing domain elements as possible witness. If this does not succeed, it extends the model with a new domain element. The difference with [4] is that our method replaces function symbols by predicate symbols, and that it relies on lemma learning during search. In this paper, we describe the search procedure without learning, because the learning rules for three-valued, geometric resolution are quite involved. They are derived from the learning rules in [8]. A detailed description can be found in [7]. We first describe the format of geometric (Kleene) formulas:

Definition 3.1. *A* geometric atom *is an atom of one of the following three forms:*

1. *An atom $p_\lambda(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables, not necessarily distinct, and $\lambda = \{\mathbf{f}\}, \{\mathbf{e}\}, \{\mathbf{t}\}, \{\mathbf{f}, \mathbf{e}\}$ or $\{\mathbf{f}, \mathbf{t}\}$.*
2. *An equality atom $x_1 \approx x_2$, with x_1, x_2 distinct variables.*
3. *An existentially quantified atom $\Sigma y\ p_\lambda(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables, not necessarily distinct, and $\lambda = \{\mathbf{e}\}$ or $\{\mathbf{t}\}$. There must be at least one occurrence of y among the x_i.*

   We will usually write $\Sigma y\ p_\lambda(x_1, \ldots, x_n, y)$, even when y does not have to be at the last position, and there can be more than one occurrence of y.

A geometric formula *is a formula of form $\Pi\overline{x}\ A_1 \oplus \cdots \oplus A_p$, where each A_i is a geometric atom with all its free variables among $\overline{x}$. It is not required that A_i contains all variables of $\overline{x}$.*

The term 'geometric atom' is slightly misleading because an object of form $\exists y\ p_\lambda(x_1, \ldots, x_n, y)$ is not an atom. We think that it is sufficiently close to an atom, so that it can still be called 'atom'. The labels in the formulas play a role similar to the signs in [14]. The search procedure is an adaptation of the procedure of [8] for classical logic. In this paper, we only describe the simplified procedure without learning.

Definition 3.2. *We assume a countably infinite set of domain elements $\mathcal{E}$. A* ground substitution Θ *is a partial function from variables to $\mathcal{E}$. If A is a geometric atom, all whose free variables are defined in Θ, then $A\Theta$ is the result of replacing each free variable x by its corresponding $x\Theta$. We call the atoms that can be obtained in this way* ground atoms*.*

As with 'geometric atom', the term 'ground atom' is not completely correct, because 'ground atoms' are not always ground, and also not always atoms, but we think they are close enough.

Definition 3.3. *An* interpretation *is a pair (E, M) in which $E \subseteq \mathcal{E}$ is a set of elements, and M is a set of ground atoms of form $p_\lambda(e_1, \ldots, e_n)$, s.t. $\lambda = \mathbf{e}$ or $\lambda = \mathbf{t}$, and all $e_i \in E$. It is not allowed that M contains a conflicting pair of form $p_\mathbf{e}(e_1, \ldots, e_n),\ p_\mathbf{t}(e_1, \ldots, e_n)$.*

An interpretation stores the ground atoms that have truth assignments different from the default value $\mathbf{f}$. Using this semantics, one can define when a ground atom is true in an interpretation. An atom of form $p_\mathbf{t}(e)$ is true in M if $p_\mathbf{t}(e)$ occurs in M. An atom of form $p_\mathbf{f}(e)$ is true in M if both of $p_\mathbf{e}(e)$ and $p_\mathbf{t}(e)$ do not occur in M. An atom of form $p_{\mathbf{f},\mathbf{t}}(e)$ is true in M if $p_\mathbf{e}(e)$ does not occur in M.

When an atom is not true in an interpretation, there are two possibilities: Either it can be made true by extending M, or it can be made true only by making M smaller. In the second case, we call the atom *in conflict* with M.

Definition 3.4. *Let (E, M) be an interpretation. Let A be a ground atom with all its elements in E. We say that A is* false *in (E, M) if one of the following holds:*

1. *A has form $e_1 \approx e_2$ and $e_1 \neq e_2$.*
2. *A has form $p_{\{\lambda\}}(e_1, \ldots, e_n)$, $\quad \lambda \neq \mathbf{f}$, and M does not contain $p_\lambda(e_1, \ldots, e_n)$.*
3. *A has form $p_\lambda(e_1, \ldots, e_n)$, $\quad \mathbf{f} \in \lambda$, and M contains an atom of form $p_\mu(e_1, \ldots, e_n)$ with $\mu \notin \lambda$.*
4. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ with $\lambda = \{\mathbf{e}\}, \{\mathbf{t}\}$, and there is no $e \in E$, s.t. $p_\lambda(e_1, \ldots, e_n, e)$ occurs in M.*

We say that A is true in (E, M) if A is not false in (E, M). We say that A is in conflict *with (E, M) if one of the following holds:*

1. *A has form $e_1 \approx e_2$ and $e_1 \neq e_2$.*
2. *A has form $p_\lambda(e_1, \ldots, e_n)$, and M contains an atom of form $p_\mu(e_1, \ldots, e_n)$ with $\mu \notin \lambda$.*

It can be seen from Definition 3.4 that an atom of form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ is never in conflict with an interpretation. This is because it is always possible to add a new element e to E, and add $p_\lambda(e_1, \ldots, e_n, e)$ to M. It is easily shown that a conflict atom is always false:

Lemma 3.5. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements occur in E. If A is in conflict with (E, M), then A is false in (E, M).*

It is also easy to show that the only way of repairing a conflict is by backtracking:

Lemma 3.6. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements occur in E. If A is in conflict with (E, M), then A is also in conflict with every interpretation (E', M'), s.t. $E \subseteq E'$ and $M \subseteq M'$.*

If an atom is false in an interpretation, but not in conflict, then it can be made true by extending the interpretation as follows:

Definition 3.7. *Let (E, M) be an interpretation. Let A be a ground atom with all its elements in E. Assume that A is false in (E, M), but not in conflict with (E, M). We say that A* extends *(E, M) into (E', M') if either*

1. *A has form $p_\lambda(e_1, \ldots, e_n)$ with $\lambda = \{\mathbf{e}\}, \{\mathbf{t}\}$, and $(E', M') = (E, M \cup \{\ p_\lambda(e_1, \ldots, e_n)\ \})$.*
2. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$ with $\lambda \in \{\mathbf{e}\}, \{\mathbf{t}\}$, and there exists an $e \in E$, s.t. $(E', M') = (\ E,\ M \cup \{\ p_\lambda(e_1, \ldots, e_n, e)\ \})$.*

3. *A has form $\Sigma y\ p_\lambda(e_1, \ldots, e_n, y)$, and there exists an $\hat{e} \notin E$, s.t. $(E', M') = (\ E \cup \{\hat{e}\},\ M \cup \{\ p_\lambda(e_1, \ldots, e_n, \hat{e})\ \})$.*

We write $(E, M) \Rightarrow_A (E', M')$, if A extends (E, M) into (E', M'),

In case A is existential, the relation $\Rightarrow_A$ is non-deterministic, because one can choose either to use an existing $e \in E$ as witness, or to create a new $\hat{e} \notin E$. In the latter case, the actual $\hat{e}$ chosen does not matter. We will always assume that there is a fixed way of obtaining a new $\hat{e} \notin E$, and that only one $\hat{e}$ will be considered.

The following lemmas states that it is always possible to extend, that atoms made true by extension will remain true, and that extension is the smallest modification that makes an atom true.

Lemma 3.8. *Let (E, M) be an interpretation. Let A be a ground atom all whose elements are in E. Assume that A is false in (E, M), but not in conflict with (E, M). Then the following hold:*

1. *There exists an interpretation (E', M'), s.t. $(E, M) \Rightarrow_A (E', M')$.*
2. *A is true in every interpretation (E', M') for which $(E, M) \Rightarrow_A (E', M')$, and in every interpretation (E'', M''), s.t. $E' \subseteq E''$ and $M' \subseteq M''$.*
3. *For every interpretation (E'', M'') with $E \subseteq E''$ and $M \subseteq M''$ in which A is true, there exists an interpretation (E', M'), s.t. $(E, M) \Rightarrow_A (E', M')$ and $E' \subseteq E''$, $M' \subseteq M''$.*

Note that part 3 implies that false atoms that are not in conflict, cannot be made true by backtracking.

Definition 3.9. *Let $F = \Pi \overline{x}\ A_1 \oplus \cdots \oplus A_p$ be a geometric formula. Let Θ be a ground substitution that is defined for all variables in $\overline{x}$. We say that F* is false *in (E, M) with Θ, if all instantiated atoms $A_i\Theta$ are false in (E, M).*

We say that F conflicts *(E, M) with Θ if each of its instantiated atoms $A_i\Theta$ is in conflict with (E, M). In that case, we call F* a conflict formula *of (E, M).*

Using extension, we define the search algorithm. It tries to extend an interpretation (E, M) into an interpretation (E', M') that makes all formulas true with every ground substitution.

At each stage, the algorithm looks for a formula F and a substitution Θ, s.t. $F\Theta$ is false in the current interpretation. If no F and Θ are found, then (E, M) is a satisfying interpretation. If F is a conflict formula, then the algorithm fails, and it backtracks. If F is not a conflict formula, then the algorithm backtracks through all possible extensions (E', M'), based on an atom A of F that is false, but not in conflict with (E, M).

Definition 3.10. *Algorithm $S_t(\mathcal{G}, E, M)$ is called with a set of geometric formulas $\mathcal{G}$ and an interpretation (E, M). It tries to extend the interpretation into an interpretation (E', M'), that makes all geometric formulas in $\mathcal{G}$ true. If no such interpretation exists, it returns $\perp$.*

If such an extension exists, it either returns an interpretation (E', M') with $E \subseteq E'$, $M \subseteq M'$, in which all $\mathcal{G}$ are true, or it does not terminate. $S_t(\mathcal{G}, E, M)$ is defined by case analysis:

MODEL: *If for all formulas $F \in \mathcal{G}$ all ground instances $F\Theta$ that use only elements in E, are true in (E, M), then $S_t(\mathcal{G}, E, M)$ returns (E, M).*

SELECT: *Otherwise, $\mathcal{G}$ contains at least one formula F, for which there exists a substitution Θ, s.t. $F\Theta$ is false in (E, M). Let $A_1, \ldots, A_m$ be the atoms in $F\Theta$ that are not in conflict with (E, M). Select an F and a Θ, for which m is minimal.*

FAIL: *If $m = 0$, then F is a conflict formula, and $S_t(\mathcal{G}, E, M)$ returns $\perp$.*

Figure 2: Propositional Geometric Formulas

(1)	$A_{\mathbf{f},\mathbf{t}}$
(2)	$A_{\mathbf{f}} \oplus B_{\mathbf{f},\mathbf{t}}$
(3)	$A_{\mathbf{f}} \oplus C_{\mathbf{f},\mathbf{t}}$
(4)	$A_{\mathbf{e}} \oplus A_{\mathbf{t}}$
(5)	$A_{\mathbf{e}} \oplus B_{\mathbf{e}}$

EXTEND: *If $m > 0$, then for every A_i, for every (E', M') with $(E, M) \Rightarrow_{A_i} (E', M')$, do the following:*

- *Assign $r = S_t(\mathcal{G}, E', M')$. If r is an interpretation, then return r.*

If we reached the end, we know that all recursive calls returned $\perp$. In that case, $S_t(\mathcal{G}, E, M)$ also returns $\perp$.

Example 3.11. *Assume that we want to refute the sequent $\#[A](B\wedge C)$, $\neg\#[A]B \vdash \perp$. Kleening the first formula results in $\#A\otimes(\neg A\oplus(\#B\otimes\#C))$. Radicalization results in $A_{\mathbf{f},\mathbf{t}}\otimes(A_{\mathbf{f}}\oplus(B_{\mathbf{f},\mathbf{t}}\otimes C_{\mathbf{f},\mathbf{t}}))$. This formula can be factored into the first three geometric formulas in Figure 2. Kleening the second formula results in $\neg\#A\oplus(A\otimes\neg\#B)$. Radicalization results in $A_{\mathbf{e}}\oplus(A_{\mathbf{t}}\otimes B_{\mathbf{e}})$. This formula can be factored into the last two formulas in Figure 2.*

We try to refute the set of formulas using algorithm S_t. We start with interpretation (E_0, M_0), defined by $E_0 = \{\}$ and $M_0 = \{\}$. Since the example is propositional, we will ignore the ground substitutions. All formulas are true in (E_0, M_0), except for the last two formulas $A_{\mathbf{e}} \oplus A_{\mathbf{t}}$ and $A_{\mathbf{e}} \oplus B_{\mathbf{e}}$.

Both formulas are not in conflict with (E_0, M_0). In the formula $A_{\mathbf{e}} \oplus A_{\mathbf{t}}$, both atoms $A_{\mathbf{e}}$ and $A_{\mathbf{t}}$ are false in (E_0, M_0) but not in conflict with (E_0, M_0). We have $(E_0, M_0) \Rightarrow_{A_{\mathbf{e}}} (E_0, M_0 \cup \{A_{\mathbf{e}}\})$, and $(E_0, M_0) \Rightarrow_{A_{\mathbf{t}}} (E_0, M_0 \cup \{A_{\mathbf{t}}\})$.

We continue search with $(E_1, M_1) = (\{\}, \{A_{\mathbf{e}}\})$. Now the first formula $A_{\mathbf{f},\mathbf{t}}$ is false in (E_1, M_1) and it is in conflict with (E_1, M_1).

We backtrack and enter the other branch, which results in $(E_2, M_2) = (\{\}, \{A_{\mathbf{t}}\})$. All formulas are true in (E_2, M_2), except for the last formula $A_{\mathbf{e}} \oplus B_{\mathbf{e}}$.

Atom $A_{\mathbf{e}}$ is in conflict with (E_2, M_2). The other atom $B_{\mathbf{e}}$ is false in (E_2, M_2), but not in conflict. We have $(E_2, M_2) \Rightarrow_{B_{\mathbf{e}}} (E_2, M_2 \cup \{B_{\mathbf{e}}\})$.

The resulting interpretation is $(E_3, M_3) = (\{\}, \{A_{\mathbf{t}}, B_{\mathbf{e}}\})$. Now the second clause $A_{\mathbf{f}} \oplus B_{\mathbf{f},\mathbf{t}}$ is false in (E_3, M_3), and it is in conflict with (E_3, M_3).

Since we have exhausted all possibilities, we have shown that the set of formulas is unsatisfiable.

This completes the discussion of the search algorithm. In its present form, the algorithm is extremely inefficient, which mostly due to the fact that it backtracks through all elements of E whenever it encounters an existential quantifier. In order to overcome these problems, the calculus has been extended with lemma learning. Details can be found in [7]. Experiments with the two-valued version of classical logic ([16]) have shown that by using learning, it is possible to obtain a practical calculus.

Figure 3: Rewriting into Geometric Formulas

$$\begin{array}{lll} A \oplus (B \otimes C) & \Rightarrow & (A \oplus B) \otimes (A \oplus C) \\ (A \otimes B) \oplus C & \Rightarrow & (A \oplus C) \otimes (B \oplus C) \\ & & \\ (\Pi x \ P[x]) \oplus A & \Rightarrow & \Pi x \ (P[x] \oplus A) \\ A \oplus \Pi x \ P[x] & \Rightarrow & \Pi x \ (A \oplus P[x]) \\ \Pi x \ (P[x] \otimes Q[x]) & \Rightarrow & (\Pi x \ P[x]) \otimes (\Pi x \ Q[x]) \end{array}$$

4 Flattening

It remains to show that sequents consisting of radicalized formulas can be transformed into sequents of geometric formulas. The transformation is mostly straightforward, and similar to standard transformations to clauses. (See [15]) The main difference is that, in order to obtain geometric formulas, function symbols have to be replaced by relation symbols. The transformations used in [1, 8] can be adopted without difficulty.

Definition 4.1. *We assume a mapping that maps every n-arity function symbol f to a unique $(n+1)$-arity predicate symbol $\overline{F}$.*

Let A be a Kleene formula that is in NNF and radicalized. An anti-Skolemization *of A is a formula that is the result of making the following replacement as long as A contains functional terms:*

Select a functional term $f(x_1, \ldots, x_n)$ in which all $x_1, \ldots, x_n$ are variables. Such a term necessarily exists. It is possible that $n = 0$.

Write A in the form $A[\ B[\ f(x_1, \ldots, x_n)\ ]\ ]$, where B is a subformula of A that contains at least one of the occurrences of $f(x_1, \ldots, x_n)$. Replace $A[\ B[\ f(x_1, \ldots, x_n)\ ]\ ]$ by $A[\ \Pi z\ \overline{F}_{\mathbf{f},\mathbf{e}}(x_1, \ldots, x_n, z) \oplus B[z]\ ]$.

After anti-Skolemization we almost have geometric logic. First rewrite the formulas, using the rewrite rules in Figure 3. If a formula has form $A \otimes B$, then it can be replaced by A and B seperately.

As long as one of the sequents contains a formula A that contains an existentially quantified subformula $\Sigma y\ P[y]$ for which $P[y]$ is not a geometric atom or does not contain y, write A in the form $A[\ \Sigma y\ P[x_1, \ldots, x_n, y]\ ]$, where $x_1, \ldots, x_n$ are the other free variables of P. Replace $A[\ \Sigma y\ P[x_1, \ldots, x_n, y]\ ]$ by $A[\ \Sigma y\ p_{\mathbf{t}}(x_1, \ldots, x_n, y)\ ]$ and $\Pi x_1 \cdots x_n\ \Pi y\ p_{\mathbf{f},\mathbf{e}}(x_1, \ldots, x_n, y) \oplus P[x_1, \ldots, x_n, y]$, using a new predicate symbol p.

Now every formula has form $\Pi \overline{x}\ (A_1 \oplus \cdots \oplus A_p)$, where $\overline{x}$ is a set of variables, and each element of $\overline{A}$ has one of the following five forms: **(1)** A geometric atom. (See Definition 3.1), **(2)** a negative equality $x_1 \not\approx x_2$ with $x_1, x_2 \in \overline{x}$, **(3)** a positive equality of form $x \approx x$ with $x \in \overline{x}$, (Positive equalities with distinct variables are geometric atoms) **(4)** $\bot$, or **(5)** $\top$. For each of these possibilities, we proceed as follows: If $x \approx x$ or $\top$ occurs among the A_i, then the formula can be completely removed. If one of the A_i has form $\bot$ or $x \not\approx x$, then A_i can be removed from the formula. If one of the A_i is a negative equality of form $x_1 \not\approx x_2$ with $x_1 \neq x_2$, it can be substituted away. The result is:

$$\Pi \overline{x} \backslash \{x_2\}\ A_1[x_1 := x_2] \oplus \cdots \oplus A_{i-1}[x_1 := x_2] \oplus A_{i+1}[x_1 := x_2] \oplus \cdots \oplus A_n[x_1 := x_2].$$

When this procedure is finished, all formulas are geometric.

Example 4.2. *Consider the sequent*

$$\left.\begin{array}{l} \forall x\ \#N(x) \\ N(0) \\ \forall x\ N(x) \rightarrow N(s(x)) \\ \neg N(s(s(0))) \end{array}\right\} \vdash \bot$$

The sequent contains two function symbols 0 *and* s*. In order to translate it into geometric format, we introduce a predicate symbol* Z *with arity* 1 *and a predicate symbol* S *with arity* 2*. The result is*

$$\left.\begin{array}{l} \Sigma y\ Z_{\mathbf{t}}(y) \\ \Pi x\ \Sigma y\ S_{\mathbf{t}}(x,y) \\ \\ \Pi x\ N_{\mathbf{f},\mathbf{t}}(x) \\ \Pi z\ Z_{\mathbf{f},\mathbf{e}}(z) \oplus N_{\mathbf{t}}(z) \\ \Pi xz\ S_{\mathbf{f},\mathbf{e}}(x,z) \oplus N_{\mathbf{f}}(x) \oplus N_{\mathbf{t}}(z) \\ \Pi z_1 z_2 z_3\ Z_{\mathbf{f},\mathbf{e}}(z_1) \oplus S_{\mathbf{f},\mathbf{e}}(z_1,z_2) \oplus S_{\mathbf{f},\mathbf{e}}(z_2,z_3) \oplus N_{\mathbf{f}}(z_3) \end{array}\right\} \vdash \bot$$

5 Conclusions

We have introduced a theorem proving procedure for PCL, that is based on geometric logic. We chose geometric logic, partially because we are used to it, and partially because we expect that it can be tuned to terminate on sequents that originate from type checking conditions. Experiments with **Geo** (an implementation of two-valued, geometric logic), have shown that geometric logic is good at terminating on unprovable goals ([16]), and reasonably good at proving goals. Although geometric logic for Kleene logic is more complicated than geometric logic for two-valued logic, it is not fundamentally different. Because of this, we expect no difficulties implementing it.

References

[1] 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.

[2] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *Logics in Artificial Intelligence (JELIA '96)*, number 1126 in LNAI. Springer, 1996.

[3] Marc Bezem and Thierry Coquand. Automating coherent logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *LPAR*, volume 3835 of *LNCS*, pages 246–260. Springer, 2005.

[4] François Bry and Sunna Torge. A deduction method complete for refutation and finite satisfiability. In Jürgen Dix, Luis Fariñas del Cerro, and Ulrich Furbach, editors, *JELIA*, volume 1489 of *LNCS*, pages 122–138. Springer Verlag, 1998.

[5] Ádám Darvas, Farhad Mehta, and Arsenii Rudich. Efficient well-definedness checking. In Alessandro Armado, Peter Baumgartner, and Gilles Dowek, editors, *International Joint Conference on Automated Reasoning (IJCAR) 2008*, volume 5195 of *LNAI*, pages 100–115. Springer Verlag, 2008.

[6] Hans de Nivelle. Classical logic with partial functions. In Jürgen Giesl and Reiner Hähnle, editors, *International Joint Conference on Automated Reasoning (IJCAR) 2010*, volume 6173 of *LNAI*, pages 203–217. Springer Verlag, 2010.

[7] Hans de Nivelle. Theorem proving for classical logic with partial functions by reduction to Kleene logic. *Journal of Logic and Computation*, pages 1–49?, 2014? (accepted for publication, a preprint can be obtained from `www.ii.uni.wroc.pl/~nivelle/publications/`.

[8] 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 *Lecture Notes in Artificial Intelligence*, pages 303–317, Seattle, USA, August 2006. Springer.

[9] William M. Farmer. Mechanizing the traditional approach to partial functions. In W. Farmer, M. Kerber, and M. Kohlhase, editors, *Proceedings of the Workshop on the Mechanization of Partial Functions (associated to CADE 13)*, pages 27–32, 1996.

[10] Reiner Hähnle. Many-valued logic, partiality, and abstraction in formal specification languages. *Logic Journal of the IGPL*, 13(4):415–433, 2005.

[11] Manfred Kerber and Michael Kohlhase. A mechanization of strong Kleene logic for partial functions. In *Automated Deduction - CADE 12*, volume 814 of *LNAI*, pages 371–385. Springer Verlag, June 1994.

[12] Sun Kim and Hantao Zhang. ModGen: Theorem proving by model generation. In Barbara Hayes-Roth and Richard Korf, editors, *Proceedings of AAAI-94*, pages 162–167, 1994.

[13] Farhad Mehta. A practical approach to partiality - a proof based approach. In Shaoying Liu and Tom Maibaum, editors, *International Conference on Formal Engineering Methods, (ICFEM)*, volume 5256 of *LNCS*, pages 238–257. Springer, 2008.

[14] Neil Murray and Erik Rosenthal. Signed formulas: A liftable meta-logic for multiple-valued logics. In Jan Komorowski and Zbigniew Raś, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, volume LNCS 689, pages 275–284, 1993.

[15] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.

[16] G. Sutcliffe. The 3rd IJCAR Automated Theorem Proving Competition. *AI Communications*, 20(2):117–126, 2007.