ISSN 1361 - 6161

THE UNIVERSITY of MANCHESTER

Computer Science
University of Manchester

A Resolution Decision Procedure for Fluted Logic

Renate A. Schmidt and Ullrich Hustadt

Department of Computer Science
University of Manchester
Technical Report Series
UMCS-00-03-1

A Resolution Decision Procedure for Fluted Logic*

Renate A. Schmidt

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
schmidt@cs.man.ac.uk

Ullrich Hustadt

Centre for Agent Research and Development
Manchester Metropolitan University
Chester Street, Manchester, UK.
U.Hustadt@doc.mmu.ac.uk

March 2000

*Refereed by: Bill Purdy

Abstract

Fluted logic is a fragment of first-order logic without function symbols in which the arguments of atomic subformulae form ordered sequences. A consequence of this restriction is that, whereas first-order logic is only semi-decidable, fluted logic is decidable. In this paper we present a sound, complete and terminating inference procedure for fluted logic. Our characterisation of fluted logic is in terms of a new class of so-called fluted clauses. We show that this class is decidable by an ordering refinement of first-order resolution and a new form of dynamic renaming, called separation.

1 Introduction

Fluted logic is of interest for a number of reasons. One of our main motivations for studying fluted logic is the continuation of the programme of characterising first-order decidability by resolution methods. There are various ways of defining decidable fragments of first-order logic. Fragments considered until the sixties usually involve some form of restriction on quantification. In prefix classes such as the Bernays-Schönfinkel class, the initially extended Ackermann class, the initially extended Gödel class the quantifier prefixes are restricted, to $\exists^*\forall^*$, $\exists^*\forall\exists^*$ and $\exists^*\forall\forall\exists^*$. In the guarded and loosely guarded fragments, which were introduced more recently, quantifiers are restricted to conditional quantifiers of the form $\exists\overline{y}G(\overline{x},\overline{y}) \wedge \varphi$ or $\forall\overline{y}G(\overline{x},\overline{y}) \rightarrow \varphi$, where $G(\overline{x},\overline{y})$ is a guard formula satisfying certain restrictions ($G(\overline{x},\overline{y})$ is an atom in the case of the guarded fragment). In Maslov's class K (more precisely, in the dual class $\overline{K}$) there is a restriction on universal quantification. Other decidable classes such as the monadic class and FO^2 are defined over predicate symbols with bounded arity. By contrast, the restriction of first-order logic which ensures decidability for fluted logic is an ordering on variables and arguments. With the exception of fluted logic, the mentioned logics have been studied in the context of resolution and superposition, see for example Joyner [17], Fermüller, Leitsch, et al. [5, 6], Bachmair, Ganzinger and Waldmann [3], de Nivelle [4], Ganzinger and de Nivelle [7], Hustadt and Schmidt [15].

Another reason for our interest in fluted logic is the relationship to non-classical logics. Extended modal logics and expressive description logics play an increasingly important role in various areas of computer science. Fluted logic may be viewed as a generalisation of modal logic, just as the guarded fragments can. The properties fluted logic is known to share with modal logics are decidability and the finite model property [20, 21, 23]. From a modal perspective an advantage of fluted logic over the guarded fragment is that relational atoms may be negated. This means that logics such as Boolean modal logic [8] and other enriched modal logics [9, 12, 13], as well as expressive description logics like $\mathcal{ALB}$ (without converse) [16], which cannot be embedded in the guarded fragment, can be embedded in fluted logic. Interestingly, translations of propositional modal formulae by both the relational translation and a variation of the functional translation (described and used in [11, 14]) are fluted formulae. This raises the question whether the results of Ohlbach and Schmidt [18, 25] can be generalised to fluted logic. The answer to this question is negative, though. Already the use of the quantifier exchange operator, which swaps existential and universal quantifiers in a non-standard fashion [18], leads to loss of soundness. A counter example is the relational translation of the second formula in the class of branching K-formulae, defined in [10, Prop. 6.5].

Historically, fluted logic arose as a byproduct of the predicate functor logic introduced by Quine [24]. Adding various combinatory operators to fluted logic defines a lattice of fluted logics, in which fluted logic is the weakest logic and first-order logic with equality is the most expressive logic. (The combinatory operators are equality, binary converse, permutation of arguments, addition of vacuous arguments, fusions of arguments, and composition of binary atoms.) In a series of papers [20, 21, 23] Purdy studies the decision problem of fluted logics in this lattice, and establishes the limit of decidability to be the boundary of the ideal generated the fluted logic with binary converse and equality [23]. This logic is the most expressive decidable logic in the lattice of fluted logics. In [22] Purdy describes an application in computational linguistics of fluted logics for modelling ordinary English.

In this paper we characterise fluted logic by a new class of clauses, called the class of *fluted clauses*. We present a decision procedure for this class which is based on an ordering refinement of resolution and an additional *separation rule*. This is a new inference rule which does dynamic renaming. It replaces a clause $C \vee D$ by two clauses $\neg A \vee C$ and $A \vee D$, where A is an atom with a newly introduced predicate symbol. The rule is sound, in general, and resolution extended by this rule remains complete, if for any set N of clauses the number of applications of separation in any derivation from N is finitely bounded. Separation is essential for our decision procedure, since it allows us to transform certain problematic fluted clauses into so-called *strongly fluted clauses*. A strongly fluted clause is a fluted clause that contains a literals which includes all the variables of the clause. When inference is restricted to such literals

(i) the number of variables in any derivable clause is finitely bounded,

in particular, the number of variables does not exceed the number of variables in the original clause set. To show termination, it is usually sufficient [17] to show in addition that

(ii) there is a bound on the depth of terms occurring in derived clauses.

Because separation introduces new predicate names during the derivation, in our case we also need to show that

(iii) there is a bound on the number of applications of the separation rule.

Exhibiting (ii) and (iii), along with verifying the deductive closure of the class of (strongly) fluted clauses are the most difficult parts of the termination proof. The difficulty can be attributed to the fact that the depth of terms can grow during the derivation, as is the case for some other solvable clausal classes, for example, those associated with Maslov's class $\overline{K}$ [5, 14].

The paper is organised as follows. Fluted logic is defined in Section 2. Section 3 gives a brief description of the general ordered resolution calculus. The class of fluted clauses is defined in Section 4. In Section 5 we specify how fluted formulae can be translated into sets of fluted clauses. The new separation rule is defined in Section 6. In Section 7 we define an ordering refinement and prove termination. We conclude with some remarks about the complexity of the class.

Throughout, our notational convention is the following: x, y, z are the letters reserved for first-order variables, s, t, u, v for terms, a, b for constants, f, g, h for function symbols, and p, q, r, P, Q, R for predicate symbols. The Greek letters φ, ψ, ϕ are reserved for formulae. A is the letter reserved for atoms, L for literals, and C, D for clauses. For sets of clauses we use the letter N.

2 Fluted Logic

Let $\mathcal{P}$ be a finite set of predicate symbols and let $X_m = \{x_1, \ldots, x_m\}$ be an *ordered* set of variables. An *atomic fluted formula* of $\mathcal{P}$ over X_i is an n-ary atom $P(x_l, \ldots, x_i)$, with $l = i - n + 1$ and $n \leq i$. *Fluted formulae* are defined inductively as follows:

1. Any atomic fluted formula over X_i is a fluted formula over X_i.
2. $\exists x_{i+1}\varphi$ and $\forall x_{i+1}\varphi$ are fluted formulae over X_i, if φ is a fluted formula over X_{i+1}.
3. Any Boolean combination of fluted formulae over X_i is a fluted formula over X_i. That is, $\varphi \to \psi$, $\neg\varphi$, $\varphi \wedge \psi$, etcetera, are fluted formulae over X_i, if both φ and ψ are.

By definition, for any formula φ, if there is a variable renaming h such that $h(\varphi)$ is a fluted formula according to the above definition then φ is a fluted formula. In this paper the assumption is that all fluted formulae are closed.

The semantics of fluted logic is defined like the semantics of first-order logic.

Three examples of fluted formulae from a linguistic or knowledge representation setting are the following. (mwmc is short for 'married couple all of whose children are married'.)

$$\forall x_1(\textsf{cheese-eater}(x_1) \leftrightarrow \exists x_2(\textsf{cheese}(x_2) \wedge \textsf{eats}(x_1, x_2)))$$
$$\forall x_1(\textsf{cheese-lover}(x_1) \leftrightarrow \forall x_2(\textsf{cheese}(x_2) \to (\textsf{eats}(x_1, x_2) \wedge \textsf{likes}(x_1, x_2))))$$
$$\forall x_1 x_2(\textsf{mwmc}(x_1, x_2) \leftrightarrow (\textsf{married}(x_1, x_2) \wedge$$
$$\forall x_3(\textsf{have-child}(x_1, x_2, x_3) \to \exists x_4 \textsf{married}(x_3, x_4))))$$

The first formula can be expressed by a multi-modal formula, while the second can only be expressed in a modal logic with an enriched language like Boolean modal logic or description logics with role negation. Because guards may only have a certain polarity the second formula does not belong to the guarded or loosely guarded fragments. The third formula is also not guarded and does not belong to Maslov's class $\overline{K}$ either. On the other hand, the formulae

$$\forall x_1 x_2(\mathsf{married}(x_1,x_2) \wedge \forall x_3(\mathsf{is\text{-}child}(x_3,x_1,x_2) \rightarrow \mathsf{doctor}(x_3))))$$
$$\forall x_1 x_2(\mathsf{married}(x_1,x_2) \wedge \exists x_3(\mathsf{have\text{-}child}(x_1,x_2,x_3) \rightarrow \exists x_4 \mathsf{married}(x_4,x_3)))$$
$$\forall x_1 x_2 x_3(\mathsf{ancestor}(x_1,x_2) \wedge \mathsf{ancestor}(x_2,x_3)) \rightarrow \mathsf{ancestor}(x_1,x_3)$$

are not fluted formulae, because in all instances the ordering of the arguments is violated in some atom.

3 Resolution

The usual definition of clausal logic is assumed. A *literal* is an atom or the negation of an atom. The former is said to be a *positive literal* and the latter a *negative literal*. In this paper *clauses* are assumed to be multisets of literals, and will be denoted by $P(x) \vee P(x) \vee \neg R(x,y)$, for example. The empty clause will be denoted by $\emptyset$. The components in the variable partition of a clause are called *variable-disjoint* or *split components*, that is, split components do not share variables. A clause which cannot be split further will be called a *maximally split clause*. The *condensation* $\mathrm{cond}(C)$ of a clause C is a minimal subclause of C which is a factor of C. We take equality of clauses (or formulae) to be equality modulo variable renaming. Two clauses (or formulae) that are equal modulo variable renaming are said to be *variants* of each other.

We say an expression is *functional* if it contains a constant or a non-nullary function symbol. Otherwise it is called *non-functional*. An expression is *shallow* if it does not contain a non-constant functional term. The set of variables of an expression E will be denoted by $\mathrm{var}(E)$.

Next, we briefly recall the definition of ordered resolution from Bachmair and Ganzinger [1, 2]. Derivations are controlled through an admissible ordering $\succ$. In the full calculus a second parameter, a selection function, may be used, but for the results of this paper it is not essential.

By definition, an ordering $\succ$ is *admissible*, if (i) it is a total well-founded ordering on the set of ground literals, (ii) for any atoms A and B, it satisfies: $\neg A \succ A$, and $B \succ A$ implies $B \succ \neg A$, and (iii) it is stable under the application of substitutions. (An ordering is said to be *liftable* if it satisfies (iii).) The multiset extension of $\succ$ provides an admissible ordering on clauses. A literal L is said to be *(strictly) maximal* with respect to a clause C if for any literal L' in C, $L' \not\succ L$ ($L' \not\succeq L$.) A literal in a clause C is said to be *eligible* if it is maximal with respect to C. An ordering is *compatible with a given complexity measure* c_L on ground literals, if $c_L \succ c_{L'}$ implies $L \succ L'$ for any two ground literals L and L'.

Let R be the resolution calculus defined by the rules of Figure 1. The completeness proof sanctions a global notion of *redundancy*, with which additional don't-care non-deterministic simplification and deletion rules can be supported. Essentially, a ground clause is redundant in a set N with respect to the ordering $\succ$ if it follows from smaller

Deduce: $\dfrac{N}{N \cup \{\text{cond}(C)\}}$ if C is a factor or resolvent of premises in N.

Delete: $\dfrac{N \cup \{C\}}{N}$ if C is redundant.

Split: $\dfrac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$ if C and D are variable-disjoint.

Resolvents and factors are computed with:

Ordered resolution: $$\frac{C \vee A_1 \quad \neg A_2 \vee D}{(C \vee D)\sigma}$$
provided (i) σ is the most general unifier of A_1 and A_2, (ii) $A_1\sigma$ is strictly maximal with respect to $C\sigma$, and (iii) $\neg A_2\sigma$ is maximal with respect to $D\sigma$.

Ordered factoring: $$\frac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$$
provided (i) σ is the most general unifier of A_1 and A_2, and (ii) $A_1\sigma$ is maximal with respect to $C\sigma$.

Figure 1: The calculus R

instances of clauses in N, and a non-ground clause is redundant in N if all its ground instances are redundant in N. For example, any tautologous clause is redundant.

A (*theorem proving*) *derivation* from a set N of clauses is a finitely branching tree with root N constructed by applications of the expansion rules. A derivation T is a *refutation* if for every path $N(= N_0), N_1, \ldots$, the clause set $\bigcup_j N_j$ contains the empty clause. A derivation T from N is called *fair* if for any path $N(= N_0), N_1, \ldots$ in the tree T, with *limit* $N_\infty = \bigcup_j \bigcap_{k \geq j} N_k$, it is the case that each clause C that can be deduced from non-redundant premises in N_∞ is contained in some set N_j.

Theorem 1 ([3]) Let N be a set of clauses and let T be a fair R-derivation from N (up to redundancy). Then, N is unsatisfiable iff for every path $N(= N_0), N_1, \ldots$, the clause set $\bigcup_j N_j$ contains the empty clause.

It should be noted that inferences with ineligible literals are not forbidden, but are provably redundant. In other words, only inferences with eligible literals need to be performed for soundness and completeness.

Strictly, the "Split" rule is inessential for the results of this paper, though it may have some computational advantages (we comment on this in the final section). However, the inclusion of splitting allows for a more concise presentation of fluted clauses.

4 Fluted Clauses

This section introduces the class of fluted clauses into which fluted formulae can be translated. Without loss of generality we consider only maximally split clauses.

In fluted clauses the arguments of literals have a characteristic form which will be described with the help of a sequence notation. $(\overline{u}_i)$ will denote a finite, possibly empty, sequence $(u_i, u_{i+1}, \ldots, u_m)$ of terms. In this paper unless specified otherwise each non-empty sequence $(\overline{u}_i)$ is assumed to end with u_m. Thus, the sequences $(\overline{u}_1), (\overline{u}_2), \ldots, (\overline{u}_m)$ are linearly ordered by (the converse of) the 'is a proper suffix of' relationship. Note that $(\overline{u}_m) = u_m$. Given that $(\overline{u}_i) = (u_i, \ldots, u_m)$,

$(\overline{u}_i, t)$	will denote the sequence $(u_i, \ldots, u_m, t)$,
$f(\overline{u}_i)$	will denote the term $f(u_i, \ldots, u_m)$,
$P(\overline{u}_i)$	will denote the atom $P(u_i, \ldots, u_m)$,
$\mathcal{C}(\overline{u}_i)$	will denote a (possibly empty) clause of literals of the form $(\neg)P(\overline{u}_i)$.

If $(\overline{u}_i)$ is the empty sequence then $f(\overline{u}_i)$, $P(\overline{u}_i)$ and $\mathcal{C}(\overline{u}_i)$ respectively denote a constant, a propositional literal and a (possibly empty) propositional clause. $(\overline{u}_i)$ is said to be the *argument sequence* of $f(\overline{u}_i)$, $P(\overline{u}_i)$ and $\mathcal{C}(\overline{u}_i)$. A sequence with n elements will be called an n-sequence. The sequence $(u_i, u_{i+1}, \ldots, u_m)$ is thus an $(m-i+1)$-sequence, and an empty sequence is a 0-sequence.

Assume m is a non-negative integer, and $X_m = \{x_1, \ldots, x_m\}$ is a set of m ordered variables. We refer to a sequence of terms $\overline{u} = (u_1, \ldots, u_n)$ as a *fluted sequence* over X_m, if the following conditions are all satisfied: (i) $n > m$, (ii) $u_1 = x_1, \ldots, u_m = x_m$, (iii) the number of variables occurring in $(u_{m+1}, \ldots, u_n)$ is m, and (iv) for every k with $m < k \leq n$, there is an i with $1 \leq i < k$ such that $u_k = f(u_i, \ldots, u_{k-1})$ for some function symbol f. The sequence $(x_1, \ldots, x_m)$ will be called the *variable prefix* of $\overline{u}$. Examples of fluted sequences are:

(a), a fluted sequence over $X_0 = \emptyset$,
$(x_1, x_2, x_3, f(x_1, x_2, x_3))$,
$(x_1, x_2, x_3, f(x_2, x_3), g(x_1, x_2, x_3, f(x_2, x_3))$,
$(x_1, x_2, f(x_1, x_2), g(f(x_1, x_2)), h(x_2, f(x_1, x_2), g(f(x_1, x_2))))$.

However, $(x_1, x_2, x_3, f(x_2, x_3))$ is not a fluted sequence, as condition (iii) is violated.

By definition, a clause C is a *fluted clause* over X_m if one of the following holds.

(FL0) C is a (possibly empty) propositional clause.

(FL1) C is not empty, $\mathrm{var}(C) = X_m$, and for any literal L in C, there is some i where $1 \leq i \leq m$ such that the argument sequence of L is

$$(x_i, x_{i+1}, \ldots, x_m).$$

(FL2) C is functional and not empty, $\mathrm{var}(C) = X_m$, and for any literal L in C, the argument sequence of L is one of

$$(x_i, x_{i+1}, \dots, x_m) \quad \text{or}$$
$$(u_j, u_{j+1}, \dots, u_n),$$

where $1 \le i \le m$ and $(u_j, u_{j+1}, \dots, u_n)$ is a suffix of some fluted sequence $\overline{u} = (u_1, \dots, u_n)$ over $\{x_k, \dots, x_m\}$, for some k with $1 \le k \le m$. $\overline{u}$ will be referred to as the *fluted sequence associated* with L. (By 4. of Lemma 1 below there can be just one fluted sequence associated with a given literal.)

(FL3) C is not empty, $\mathrm{var}(C) = X_{m+1}$, and for any literal L in C, the argument sequence of L is either

$$(x_1, x_2, \dots, x_m) \quad \text{or}$$
$$(x_i, \dots, x_m, x_{m+1}),$$

where $1 \le i \le m$.

A fluted clause will be called a *strongly fluted clause* if it is either ground or has a literal which contains all the variables of the clause.

It may be helpful to consider some examples. The clause

$$P(x_1, x_2, x_3, x_4, x_5) \lor Q(x_1, x_2, x_3, x_4, x_5) \lor \neg R(x_4, x_5) \lor S(x_5).$$

satisfies the scheme (FL1), and is defined over five variables. Examples of fluted clauses of type (FL3) which are defined over two (!) variables are:

$$Q(x_1, x_2) \lor \neg P(x_1, x_2, x_3) \lor \neg R(x_2, x_3) \lor S(x_3)$$
$$Q(x_1, x_2) \lor \neg R(x_2, x_3) \lor S(x_3)$$

The following are fluted clauses of type (FL2), where $(\overline{x}_1) = (x_1, \dots, x_4)$.

$$R(x_2, f(x_1, x_2)) \lor S(f(x_1, x_2))$$
$$Q(x_1, x_2) \lor R(x_2) \lor P(x_1, x_2, f(x_1, x_2)) \lor R(x_2, f(x_1, x_2)) \lor S(f(x_1, x_2))$$
$$Q(x_3, x_4) \lor R(x_4) \lor P(x_4, g(\overline{x}_1), f(x_4, g(\overline{x}_1))) \lor R(f(x_4, g(\overline{x}_1)))$$
$$Q(\overline{x}_1) \lor P(f(\overline{x}_1, h(\overline{x}_1)), g(\overline{x}_1, h(\overline{x}_1), f(\overline{x}_1, h(\overline{x}_1)))$$
$$\lor R(x_4, h(\overline{x}_1), g'(x_4, h(\overline{x}_1)))$$

A few remarks are in order. First, the non-functional subclause of a (FL2)-clause will be denoted by ∇. Note that ∇ satisfies (FL1), in other words, clauses of the form (FL1) are building blocks of (FL2)-clauses.

Second, clauses of the form (FL3) are defined to be fluted clauses over m variables, even though they contain $m+1$ variables. This may seem a bit strange, but this definition ensures a direct association of fluted formulae over m variables to fluted clauses over m variables.

Third, no fluted clause can simultaneously satisfy any two of (FL0), (FL1), (FL2) and (FL3).

Fourth, using the previously introduced notation a schematic description of the non-propositional shallow clauses (FL1) and (FL3) is:

$$\mathcal{C}(\overline{x}_1) \vee \mathcal{C}(\overline{x}_2) \vee \ldots \vee \mathcal{C}(x_m) \quad (= \nabla)$$
$$\mathcal{C}(\overline{x}_1) \vee \mathcal{C}(\overline{x}_1, x_{m+1}) \vee \mathcal{C}(\overline{x}_2, x_{m+1}) \vee \ldots \vee \mathcal{C}(x_m, x_{m+1}) \vee \mathcal{C}(x_{m+1})$$

Fifth, strongly fluted clauses have special significance in connection with termination of resolution, particularly with respect to the existence of a bound on the number of variables in any clause. Under the refinement we will use the eligible literals are literals which contain all the variables of the clause. So, the number of variables in resolvents of strongly fluted premises will always be less than or equal to the number of variables in any of the parent clauses.

The next results give some properties of fluted sequences and strongly fluted clauses.

Lemma 1 Let $\overline{u}$ be a fluted sequence over X_m. Then:

1. There is an element u_k of $\overline{u}$ such that $u_k = f(u_1, \ldots, u_{k-1})$, for some f.
2. If u_n is last element of $\overline{u}$ then $\text{var}(u_n) = X_m$.
3. $\overline{u}$ is uniquely determined by its last element.
4. If $(u_j, u_{j+1}, \ldots, u_n)$ is a suffix of $\overline{u}$ then $(u_1, \ldots, u_{j-1})$ is uniquely determined by $(u_j, u_{j+1}, \ldots, u_n)$.

Proof. If X_m is empty then the results hold trivially. Let's assume that $X_m \neq \emptyset$. The definition of a fluted sequence is such that at least the argument u_{l-1} preceding any functional u_l in $\overline{u}$ is an argument of u_l (condition (iv)). More precisely, functional elements propagate at least the preceding element to the next element. This implies that the functional elements preceding a functional element u_l occur all u_l. By condition (iii), there is at least one element, u_k say, which includes also the variable prefix. This proves 1. Thus all elements following this element u_k will include the variable prefix, in particular, the last element will also. This proves 2. It also follows that u_n is a nested term from which $\overline{u}$ may be constructed (property 3.). Property 4. is a consequence of 3. □

By the definition of (FL2)-clauses:

Lemma 2 Let L be any literal of a (FL2)-clause defined over X_m. Then, all occurrences of variable sequences in L are suffixes of $(x_1, \ldots, x_m)$.

Lemma 3 Let C be a fluted clause over m variables. C is strongly fluted iff

1. C satisfies exactly one of the conditions (FL0), (FL1), (FL2), or
2. C satisfies condition (FL3), and it contains a literal with $m+1$ variables.

Proof. 1. is easy to see. 2. follows from the previous lemma. □

In other words, with the exception of certain (FL3)-clauses all fluted clauses include at least one literal which contains all the variables of the clause.

5 From Fluted Formulae to Fluted Clauses

Our transformation of fluted formulae into clausal form employs a standard renaming technique, known as structural transformation or renaming, see for example [19].

Define the polarity of (occurrences of) first-order subformulae as usual: Any occurrence of a proper subformula of an equivalence has *zero polarity*. For occurrences of subformulae not below a '$\leftrightarrow$' symbol, an occurrence of a subformula has *positive polarity* if it is one inside the scope of an even number of (explicit or implicit) negations, and it has *negative polarity* if it is one inside the scope of an odd number of negations. Let $\mathrm{Pos}(\varphi)$ be the set of positions of a first-order formula φ. If λ is a position in φ, then $\varphi|_\lambda$ denotes the subformula of φ at position λ and $\varphi[\psi \mapsto \lambda]$ is the result of replacing $\varphi|_\lambda$ at position λ by ψ.

Structural transformation associates with each element λ of $\Lambda \subseteq \mathrm{Pos}(\varphi)$ a predicate symbol Q_λ and a literal $Q_\lambda(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are the free variables of $\varphi|_\lambda$, the symbol Q_λ does not occur in φ and two symbols Q_λ and $Q_{\lambda'}$ are equal only if $\varphi|_\lambda$ and $\varphi|_{\lambda'}$ are equivalent formulae. Let

$$\begin{aligned}
\mathrm{Def}^+_\lambda(\varphi) &= \forall x_1 \ldots x_n (Q_\lambda(x_1, \ldots, x_n) \rightarrow \varphi|_\lambda) \quad \text{and} \\
\mathrm{Def}^-_\lambda(\varphi) &= \forall x_1 \ldots x_n (\varphi|_\lambda \rightarrow Q_\lambda(x_1, \ldots, x_n)).
\end{aligned}$$

The *definition* of Q_λ is the formula

$$\mathrm{Def}_\lambda(\varphi) = \begin{cases} \mathrm{Def}^+_\lambda(\varphi) & \text{if } \varphi|_\lambda \text{ has positive polarity} \\ \mathrm{Def}^-_\lambda(\varphi) & \text{if } \varphi|_\lambda \text{ has negative polarity} \\ \mathrm{Def}^+_\lambda(\varphi) \wedge \mathrm{Def}^-_\lambda(\varphi) & \text{otherwise.} \end{cases}$$

The corresponding clauses are called *definitional clauses.* Define $\mathrm{Def}_\Lambda(\varphi)$ inductively by:

$$\begin{aligned}
\mathrm{Def}_\emptyset(\varphi) &= \varphi \quad \text{and} \\
\mathrm{Def}_{\Lambda \cup \{\lambda\}}(\varphi) &= \mathrm{Def}_\Lambda(\varphi[Q_\lambda(x_1, \ldots, x_n) \mapsto \lambda]) \wedge \mathrm{Def}_\lambda(\varphi),
\end{aligned}$$

where λ is maximal in $\Lambda \cup \{\lambda\}$ with respect to the prefix ordering on positions.

For any first-order formula φ, $\mathrm{Def}_\Lambda(\varphi)$ will denote the *definitional form* obtained by introducing new names for subformulae at positions in Λ. Usually Λ is the subset of all positions of subformulae which are non-atomic or non-literal.

Theorem 2 Let φ be a first-order formula. For any $\Lambda \subseteq \text{Pos}(\varphi)$,

1. φ is satisfiable iff $\text{Def}_\Lambda(\varphi)$ is satisfiable, and
2. $\text{Def}_\Lambda(\varphi)$ can be computed in polynomial time.

In this paper we assume the clausal form of a first-order formula φ, written $\text{Cls}(\varphi)$, is computed by transformation into conjunctive normal form, outer Skolemisation, and clausifying the Skolemised formula.

By introducing new literals for each non-literal subformula position, any given fluted formula can be transformed into a set of strongly fluted clauses.

Lemma 4 Let φ be any fluted formula. If Λ contains all non-literal subformula positions of φ then $\text{ClsDef}_\Lambda(\varphi)$ is a set of strongly fluted clauses (provided the newly introduced literals have the form $(\neg)Q_\lambda(\overline{x}_i)$).

Proof. Definitional clauses in $\text{Def}_\Lambda(\varphi)$ have one of the following forms, where $\lambda \in \Lambda$ and $\varphi|_\lambda$ is the subformula at position λ. (Below, if $\overline{x}_1$ is an empty sequence then $\overline{x}_i$ is empty as well.)

1. $(\neg)Q_\lambda(\overline{x}_1) \vee (\neg)Q_1(\overline{x}_1)\ [\vee\ (\neg)Q_2(\overline{x}_i)]$	if $\varphi\|_\lambda$ is a Boolean subformula
2. $(\neg)Q_\lambda(\overline{x}_1) \vee (\neg)Q_1(\overline{x}_i)$	
3. $(\neg)Q_\lambda(\overline{x}_1) \vee (\neg)Q_1(\overline{x}_1, f(\overline{x}_1))$	if $\varphi\|_\lambda = \exists x_{m+1}\psi$
4. $(\neg)Q_\lambda(\overline{x}_1) \vee (\neg)Q_1(\overline{x}_1, x_{m+1})$	if $\varphi\|_\lambda = \forall x_{m+1}\psi$
5. $(\neg)Q_\epsilon$	for the root position ϵ

1. and 2. are (FL1)-clauses, 3. is a (FL2)-clause, 4. is a (FL3)-clause, and 5. is a (FL0)-clause. □

Transforming any given fluted formula into a set of *fluted* clauses requires the introduction of new symbols for all quantified subformulae.[1]

Lemma 5 Let φ be any fluted formula over m ordered variables. If Λ contains at least the positions of any subformulae $\exists x_{i+1}\psi$, $\forall x_{i+1}\psi$, then $\text{ClsDef}_\Lambda(\varphi)$ is a set of fluted clauses (again, provided the new literals have the form $(\neg)Q_\lambda(\overline{x}_i)$).

Proof. Consider a subformula $\exists x_{i+1}\psi$ of φ such that ψ is quantifier-free. Assume $\exists x_{i+1}\psi$ occurs at position λ and is a fluted formula over i ordered variables. Assume further the

[1]More generally, it requires at least the introduction of new symbols for all positive occurrences of universally quantified subformulae, all negative occurrences of existentially quantified subformulae, and all quantified subformulae with zero polarity. But then inner Skolemisation needs to be used, first Skolemising the deepest existential formulae.

number of free variables of $\exists x_{i+1}\psi$ is $n \geq 0$. We introduce a new predicate symbol Q_λ with arity n and let

$$\begin{aligned}\mathrm{Def}^+_\lambda(\varphi) &= \forall x_l \dots x_i \ (Q_\lambda(x_l, \dots, x_i) \rightarrow \exists x_{i+1}\psi) \quad \text{and} \\ \mathrm{Def}^-_\lambda(\varphi) &= \forall x_l \dots x_i \ (\exists x_{i+1}\psi \rightarrow Q_\lambda(x_l, \dots, x_i)),\end{aligned}$$

where $l = i-n+1$ if $n > 0$, and $l(= i) = 0$, otherwise. It is routine to verify that the clausal form of $\mathrm{Def}^+_\lambda(\varphi)$ consists of (FL2)-clauses, which are strongly fluted and contain n variables. For $\mathrm{Def}^-_\lambda(\varphi)$ the clausal form consists of (FL3)-clauses with $n+1$ variables. It is here where strong flutedness cannot be guaranteed. For example, if $\psi = P_1(x_1, x_2, x_3) \vee P_2(x_2, x_3)$ then the clausal form of $\mathrm{Def}^-_\lambda(\varphi)$ is

$$\begin{aligned}&Q_\lambda(x_1, x_2) \vee \neg P_1(x_1, x_2, x_3) \\ &Q_\lambda(x_1, x_2) \vee \neg P_2(x_2, x_3)\end{aligned}$$

of which the latter does not contain a literal which contains all the variables of the clause, and is thus only fluted, not strongly fluted. □

6 Separation

The motivation for introducing separation is that the class of fluted clauses is not closed under resolution. In particular, resolvents of non-strongly fluted (FL3)-clauses are not always fluted and can cause (potentially) unbounded variable chaining across literals. This is illustrated by considering resolution between $P_1(x_1, x_2) \vee Q_1(x_2, x_3) \vee R(x_2, x_3)$ and $\neg R(x_1, x_2) \vee P_2(x_1, x_2) \vee Q_2(x_2, x_3)$, which produces the resolvent $P_1(x_1, x_2) \vee Q_1(x_2, x_3) \vee P_2(x_2, x_3) \vee Q_2(x_3, x_4)$. We note that it contains four variables, whereas the premises each contain only three variables. The class of strongly fluted clauses is also not closed under resolution. Fortunately, however, inferences with two strongly fluted clauses always produce fluted clauses, and non-strongly fluted clauses are what we call separable and can be restored to strongly fluted clauses.

Consider the resolvent $C = P(x_1, x_2) \vee P(x_2, x_3)$ of the strongly fluted clauses $P(x_1, x_2) \vee R(x_1, x_2, x_3)$ and $\neg R(x_1, x_2, x_3) \vee P(x_2, x_3)$. C is a fluted clause of type (FL3), but it is not strongly fluted, as none of its literals contains all the variables of the clause. Consequently, the literals are incomparable under an admissible ordering (in particular, a liftable ordering), because the literals have a common instance, for example $C\{x_1 \mapsto a, x_2 \mapsto a, x_3 \mapsto a\} = P(a, a) \vee P(a, a)$. The 'culprits' are the variables x_1 and x_3. Because they do not occur together in any literal, C can be separated and replaced by the following two clauses, where q is a new predicate symbol.

$$\begin{aligned}\neg q(x_2) &\vee P(x_1, x_2) \\ q(x_2) &\vee P(x_2, x_3)\end{aligned}$$

The first clause is of type (FL1) (and thus strongly fluted) and the second is a strongly fluted clause of type (FL3). Similarly, separation will replace the clauses $P_1(x_1, x_2) \vee$

$Q_1(x_2, x_3) \vee \neg R(x_2, x_3)$ and $R(x_1, x_2) \vee P_2(x_1, x_2) \vee Q_2(x_2, x_3)$ by the set:

$$\begin{aligned}
&\neg q_1(x_2) \vee P_1(x_1, x_2)\\
&\ \ q_1(x_2) \vee Q_1(x_2, x_3) \vee \neg R(x_2, x_3)\\
&\neg q_2(x_2) \vee R(x_1, x_2) \vee P_2(x_1, x_2)\\
&\ \ q_2(x_2) \vee Q_2(x_2, x_3)
\end{aligned}$$

Under our resolution refinement (to be defined formally in the next section) the literals resolved upon are those with maximal arity. Thus, assuming $\neg R(x_2, x_3) \succ Q_1(x_2, x_3)$ and $R(x_1, x_2) \succ P_2(x_1, x_2)$ a resolvent is the (strongly) fluted clause:

$$q_1(x_2) \vee Q_1(x_2, x_3) \vee R(x_2, x_3) \vee P_2(x_2, x_3) \vee \neg q_2(x_3).$$

In the remainder of this section we will formally define separation and consider under which circumstances soundness and completeness hold. In the next section we will show how separation can be used to stay within the class of fluted clauses.

Let C be an arbitrary (not necessarily fluted) clause. C is *separable* if it can be partitioned into two non-empty subclauses D_1 and D_2 such that $\mathrm{var}(D_1) \not\subseteq \mathrm{var}(D_2)$ and $\mathrm{var}(D_2) \not\subseteq \mathrm{var}(D_1)$. For example, the clauses $P(x_1, x_2) \vee Q(x_2, x_3)$ and $P(x_1) \vee Q(x_2)$ are separable, but $P(x_1, x_2) \vee Q$ and $P(x_1, x_2) \vee Q(x_2, x_3) \vee R(x_1, x_3)$ are not. (The last clause is not fluted.)

Theorem 3 Let $C \vee D$ be a separable clause such that $\mathrm{var}(C) \not\subseteq \mathrm{var}(D)$, $\mathrm{var}(D) \not\subseteq \mathrm{var}(C)$, and $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1, \ldots, x_n\}$ for $n \geq 0$. Let q be a fresh predicate symbol with arity n (q does not occur in N). Then, $N \cup \{C \vee D\}$ is satisfiable iff $N \cup \{\neg q(x_1, \ldots, x_n) \vee C,\ q(x_1, \ldots, x_n) \vee D\}$ is satisfiable.

Proof. The 'if' direction is clear since

$$((\neg q(x_1, \ldots, x_n) \vee C) \wedge (q(x_1, \ldots, x_n) \vee D)) \rightarrow (C \vee D).$$

For the 'only if' direction, we show that any interpretation I in which $C \vee D$ is true can be extended conservatively to an interpretation I' in which the new clauses are true.

Let $X = \{x_1, \ldots, x_n\}$ be the set of all common variables of C and D. Let $\overline{s}$ be an arbitrary n-sequence of ground terms. Let $\Theta(\overline{s})$ be the set of all ground substitutions σ such that the restriction of σ to the variables in X is $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$. Assign $q(s_1, \ldots, s_n)$ true in I' iff $C\sigma$ is true in I (and I') for all ground substitutions $\sigma \in \Theta(\overline{s})$. It follows that both clauses

$$\neg q(x_1, \ldots, x_n)\sigma \vee C\sigma \quad \text{and} \quad q(x_1, \ldots, x_n)\sigma \vee D\sigma$$

are true in I': If $q(x_1, \ldots, x_n)\sigma$ is true in I' then, by definition, $C\sigma$ is true in I' and hence both clauses are true in I'. If $q(x_1, \ldots, x_n)\sigma$ is false in I' then we can show that $D\theta$ is true in I, and I', for every ground substitution θ in $\Theta(\overline{s})$, and both the introduced clauses are true in I'.

To show that $D\theta$ is true in I, and I', for every ground substitution θ in $\Theta(\overline{s})$, assume the opposite. That is, assume there is a substitution $\theta \in \Theta(\overline{s})$ such that $D\theta$ is false in I. Since $q(x_1, \ldots, x_n)\sigma$ is false in I', by definition, there is a substitution θ' such that $C\theta'$ is false in I. Now let θ'' be the substitution $\theta'|_{\mathrm{var}(C)} \cup \theta|_{\mathrm{var}(D)}$. Note that θ' and θ coincide on X, and that θ'' is in $\Theta(\overline{s})$. Since $\theta''|_{\mathrm{var}(C)} = \theta'|_{\mathrm{var}(C)}$ and $\theta''|_{\mathrm{var}(D)} = \theta|_{\mathrm{var}(D)}$ both $C\theta''$ and $D\theta''$ are false in I. This contradicts the assumption that I is an interpretation in which $C \vee D$ is true. □

On the basis of this theorem we can define the following replacement rule:

Separate: $$\frac{N \cup \{C \vee D\}}{N \cup \{\neg q(x_1, \ldots, x_n) \vee C,\ q(x_1, \ldots, x_n) \vee D\}}$$

provided (i) $C \vee D$ is separable such that $\mathrm{var}(C) \not\subseteq \mathrm{var}(D)$ and $\mathrm{var}(D) \not\subseteq \mathrm{var}(C)$, (ii) $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1, \ldots, x_n\}$ for $n \geq 0$, and (iii) q does not occur in N, C or D.

C and D will be referred to as the *separation components* of $C \vee D$.

Lemma 6 The replacements of a separable clause C each contain less variables than C.

Proof. By side condition (i) of the “Separate” rule. □

Even though it is possible to define an ordering under which the replacement clauses are strictly smaller than the original clause, and consequently, $C \vee D$ is redundant in $N \cup \{\neg q(x_1, \ldots, x_n) \vee C,\ q(x_1, \ldots, x_n) \vee D\}$, in general, “Separate” is not a simplification rule in the sense of Bachmair-Ganzinger. Nevertheless, we can prove the following.

Theorem 4 Let $\mathsf{R}^{\mathrm{sep}}$ denote the extension of R with the separation inference rule. Let N be a set of clauses and let T be a fair $\mathsf{R}^{\mathrm{sep}}$-derivation from N such that separation is applied only finitely often in any path of T. Then N is unsatisfiable iff for every path $N(= N_0), N_1, \ldots$, the clause set $\bigcup_j N_j$ contains the empty clause.

Proof. Let $N(= N_0), N_1, \ldots$ be a path of T such that the empty clause is not in $\bigcup_j N_j$. We show that N is satisfiable. Since separation is applied only finitely often in this path, there is an index k such that separation is not applied to any set of clauses N_i with $i \geq k$. By induction we can show that for every i with $1 \leq i \leq k$, N is satisfiable if N_i is satisfiable using Theorem 3 and corresponding results for the inference rules of R. That is, N is satisfiable if N_k is satisfiable. For the remaining (possibly infinite) part $N_k, N_{k+1}, \ldots$ it follows from Theorem 1 that N_k is satisfiable since $\bigcup_{j \geq k} N_j$ does not contain the empty clause. Thus, N is satisfiable.

Now suppose that for every path $N(= N_0), N_1, \ldots, \bigcup_j N_j$ contains the empty clause. We show that N is unsatisfiable. For each path $N_0, N_1, \ldots$ there exists an index k such that separation is not applied to any set of clauses N_i with $i \geq k$, This determines a clause set N_k for each path of T. Since the derivation tree is finitely branching, there are only finitely many such clause sets $N'_0, \ldots, N'_n$. Each of these clause sets induces a subtree T'_i,

$1 \le i \le n$, of T. Deleting these subtrees from T leaves a tree T' of finite depth such that each leave of T' is equal to one of the N_i', $1 \le i \le n$. For every path $N_i'(= N_0), N_1, \ldots$ in every subtree T_i', $\bigcup_j N_j$ contains the empty clause. Since T_i' is generated by R, it follows by Theorem 1 that N_i' is unsatisfiable. By induction on the number of paths in T' we can now show that the unsatisfiability of every N_i', $1 \le i \le n$, implies the unsatisfiability of N using Theorem 3 and corresponding results for the inference rules of R. □

More generally, this theorem holds also if R^{sep} is based on ordered resolution (or superposition) with selection.

By Lemma 3 separable fluted clauses have the form

$$\mathcal{C}(\overline{x}_1) \vee \mathcal{C}(\overline{x}_i, x_{m+1}) \vee \ldots \vee \mathcal{C}(x_m, x_{m+1}) \vee \mathcal{C}(x_{m+1}), \tag{1}$$

where $(\overline{x}_1)$ is a non-empty m-sequence, $\mathcal{C}(\overline{x}_1)$ is not empty, and i is the smallest integer $1 < i \le m$ such that $\mathcal{C}(\overline{x}_i, x_{m+1})$ is not empty.

Let sep be a mapping from separable fluted clauses of the form (1) to sets of clauses defined by

$$\begin{aligned}\mathrm{sep}(C) = \{&\neg q(\overline{x}_i) \vee \mathcal{C}(\overline{x}_1),\\ &q(\overline{x}_i) \vee \mathcal{C}(\overline{x}_i, x_{m+1}) \vee \ldots \vee \mathcal{C}(x_m, x_{m+1}) \vee \mathcal{C}(x_{m+1})\}\end{aligned}$$

where q is a fresh predicate symbol uniquely associated with C and all its variants. Further, let $\mathrm{sep}(N) = \bigcup\{\mathrm{sep}(C) \mid C \in N\}$. For example:

$$\mathrm{sep}(P(x_1, x_2) \vee Q(x_2, x_3)) = \{\neg q(x_2) \vee P(x_1, x_2),\ q(x_2) \vee Q(x_2, x_3)\}.$$

Lemma 7 The separation of a separable fluted clause (1) is a set of strongly fluted clauses.

Lemma 8 For fluted clauses a separation inference step can be performed in linear time.

Proof. Let n be the maximal arity of any predicate symbol in a given clause C of the form (1). We need one traversal and cn steps for gathering the ordered sequence $(x_1, ..., x_{m+1})$ of all variables from the clause (c is a constant). Exploiting the particular form of separable clauses it requires just one additional traversal of C to extract the separation components. □

7 Termination

In this section we define a minimal resolution calculus R^{sep} and prove that it provides a decision procedure for fluted logic.

The ordering $\succ$ of R^{sep} is required to be any admissible ordering compatible with the following complexity measure. Let $\succ_s$ denote the proper superterm ordering. Define the complexity measure of any literal L by $c_L = (\mathrm{ar}(L), \max(L), \mathrm{sign}(L))$, where $\mathrm{ar}(L)$ is the arity (of the predicate symbol) of L, $\max(L)$ is a $\succ_s$-maximal term occurring in L, and $\mathrm{sign}(L) = 1$, if L is negative, and $\mathrm{sign}(L) = 0$, if L is positive. The ordering on the complexity measures is given by the lexicographic combination of $>$, $\succ_s$, and $>$ (where $>$ is the usual ordering on the non-negative integers).

Let $\mathsf{R}^{\mathrm{sep}}$ be any calculus in which (i) derivations are generated by strategies applying "Delete", "Split", "Separate", namely, $N \cup \{C\}/N \cup \mathrm{sep}(C)$, and "Deduce" in this order, (ii) no application of "Deduce" with identical premises and identical consequence may occur twice on the same path in derivations, and (iii) the ordering is based on $\succ$, defined above.

Now we address the question as to whether the class of fluted clauses is closed under $\mathsf{R}^{\mathrm{sep}}$-inferences.

Lemma 9 A factor of a strongly fluted clause C is again a strongly fluted clause of the same type as C.

Proof. Let $C = C' \vee A_1 \vee A_2$ be a strongly fluted clause and let A_1 and A_2 be eligible literals. In all cases (FL0)–(FL3), A_1 and A_2 will be syntactically identical. □

In fact, any (unordered) factor of a strongly fluted clause C is again a strongly fluted clause of the same type.

The next lemma is the most important technical result. In it we will make use of the notion of a *near fluted sequence*. By this we mean a sequence which satisfies the conditions of a fluted sequence, except possibly condition (iii). In other words, a near fluted sequence $\overline{v}$ is a fluted sequence if the variables occurring in the functional elements are exactly the variables which occur in the variable prefix of $\overline{v}$.

Lemma 10 Let $C = C' \vee A_1$ and $D = \neg A_2 \vee D'$ be (FL2)-clauses. Suppose A_1 and $\neg A_2$ are eligible literals in C and D, respectively, and suppose σ is the most general unifier of A_1 and A_2. Then:

1. $C\sigma$, $D\sigma$ and $C\sigma \vee D\sigma$ are (FL2)-clauses.

2. For any functional literal $L\sigma$ in $C\sigma \vee D\sigma$, the fluted sequence associated with $L\sigma$ is the σ-instance of a fluted sequence $\overline{v}$ associated with some literal L' in $C \vee D$.

Proof. (a) Both A_1 and A_2 are non-functional. Then A_1 and A_2 are variants of an atom $P(\overline{x}_1)$, for some P and $(\overline{x}_1)$. Hence σ is a variable renaming, which preserves the positions of the variables in A_1 and A_2. By this we mean that any variable occurring in the ith last argument of A_1 (A_2) is only instantiated with the variable occurring in the ith last argument of A_2 (A_1). We say σ is a *variable-position preserving* renaming. Both $C\sigma$ and $D\sigma$ and hence their union $C\sigma \vee D\sigma$ are (FL2)-clauses. This proves 1. for this case.

For 2. it is not difficult to see that $\overline{v}\sigma$ is a fluted sequence associated with $L\sigma$, with $L\sigma \in C\sigma \vee D\sigma$, given that $\overline{v}$ is a fluted sequence associated with L, and σ is a variable-position preserving renaming.

(b) A_1 is non-functional and A_2 is functional. Suppose $A_1 = P(\overline{x}_1)$, $A_2 = P(\overline{u}_1)$, and both $(\overline{x}_1)$ and $(\overline{u}_1)$ are m-sequences. Then $\mathrm{var}(D) = \mathrm{var}((\overline{u}_1)) = \mathrm{var}(u_m)$, by Lemma 1. If $\mathrm{var}(u_m) = \emptyset$ then $(\overline{u}_1) = (a)$ where a is a constant, and the result follows easily. We will thus assume that $(\overline{u}_1)$ contains variables. Let σ be a most general unifier of A_1 and A_2. Let u_k be the first argument of A_2 which is not a variable. (Note that $1 \leq k \leq |\mathrm{var}(D)|$, i.e. D may contain more than k variables, since the maximal literal has maximal arity but may not contain all the variables of the clause.) Then the restriction of σ to $\{u_1, \ldots, u_{k-1}\}$ is a variable-position preserving renaming. Restricted to the variables of D, σ is only a variable renaming. This implies $D\sigma$ is still a (FL2)-clause and 2. holds for every literal in $D\sigma$. We now prove that $C\sigma$ is also a (FL2)-clause and satisfies 2.

First consider the $\nabla\sigma$ subclause of $C\sigma$. (Remember no function symbols occur in the ∇ subclause of a (FL2)-clause, and ∇ is a (FL1)-clause.) ∇ is not empty as $A_1 \in \nabla$. Assume the fluted sequence associated with A_2 is $\overline{v}$. Then $\overline{v}\sigma$ is a variant of $\overline{v}$ and hence $\overline{v}\sigma$ is the fluted sequence associated with $A_2\sigma$. Since $A_1\sigma = A_2\sigma$, it follows $\overline{v}\sigma$ is also the fluted sequence associated with $A_1\sigma$. Because the argument sequences of all literals in ∇ are suffixes of $(\overline{x}_1)$ (else, A_1 would not have maximal arity), the argument sequences of $\nabla\sigma$ are suffixes of $(\overline{u}_1)\sigma$, and hence, they are suffixes of $\overline{v}\sigma$. In other words, $\overline{v}\sigma$ is also the fluted sequence associated with each literal in $\nabla\sigma$. This proves 2. It also implies that $\nabla\sigma$ is a (FL2)-clause.

It remains to consider $C\backslash\nabla$. We show that, for any literal L in $C\backslash\nabla$, the σ-instance of a fluted sequence associated with L is a fluted sequence, moreover, it is the fluted sequence associated with $L\sigma$. Assume L is an arbitrary literal in $C\backslash\nabla$ and its associated fluted sequence is $\overline{v} = (v_1, \ldots, v_n)$. There are two cases: (i) the variable prefix of $\overline{v}$ is a suffix of $(\overline{x}_1)$, or (ii) $(\overline{x}_1)$ is a proper suffix of the variable prefix of $\overline{v}$. We will only consider case (ii), as case (i) can be argued similarly.

So assume $\overline{v}$ has the form

$$(y_1, \ldots, y_i, x_1, \ldots, x_m, v_j, \ldots, v_n),$$

that is, v_j is the first element of $\overline{v}$ which is not a variable. In order to simplify the presentation of the remainder of the proof we will assume that $\sigma = \{x_1 \mapsto u_1, \ldots, x_m \mapsto u_m\}$. Due to this assumption, σ is an idempotent unifier and may be decomposed into m bindings $\theta_1, \ldots, \theta_m$, where θ_i is the binding which affects position i of $(\overline{x}_1)$, i.e. $\theta_i = \{x_i \mapsto u_i\}$. Evidently, as each of $\theta_1 \ldots \theta_{k-1}$ are variable renamings, and preserve the positions of variables, $\overline{v}\theta_1 \ldots \theta_{k-1}$ is the fluted sequence for $L\theta_1 \ldots \theta_{k-1}$. If θ is the abbreviation for $\theta_1 \ldots \theta_{k-1}$ then $\overline{v}\theta_1 \ldots \theta_{k-1}$ is a sequence

$$(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}, x_k \ldots, x_m, v_j\theta, \ldots, v_n\theta).$$

$\theta_k = \{x_k \mapsto u_k\}$ is the first binding which instantiates a variable with a functional term. Hence $x_k\theta_k = u_k$. The sequence

$$(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}, x_k\theta_k)$$

is a near fluted sequence with variable prefix

$$\overline{y} = (y_1, \ldots, y_i, x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}).$$

This is a consequence of the fact that u_k is a functional term and its argument sequence is a suffix of $x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}$. Similarly, each of the following are near fluted sequences, with the same variable prefix $\overline{y}$.

$$\begin{aligned}
&(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}, x_k\theta_k, x_{k+1}\theta_{k+1}) \\
&\vdots \\
&(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_{k-1}\theta_{k-1}, x_k\theta_k, \ldots, x_m\theta_m)
\end{aligned}$$

The final step is to consider the sequences

$$\begin{aligned}
&(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_m\theta_m, v_j\sigma) \\
&\vdots \\
&(y_1, \ldots, y_i, x_1\theta_1, \ldots, x_m\theta_m, v_j\sigma \ldots, v_n\sigma)
\end{aligned}$$

Each is a near fluted sequence, because each $v_j\sigma, \ldots, v_n\sigma$ is a functional term of the required form. For example, $v_j\sigma = f(v_l\sigma, \ldots, v_{j-1}\sigma)$, because $v_j = f(v_l, \ldots, v_{j-1})$, where $1 \leq l < j$. By 1. of Lemma 1 the original fluted sequence $\overline{v}$ contains a functional term w with the variable prefix of $\overline{v}$ as arguments. It is not difficult to see that $w\sigma$ contains the variable prefix of $\overline{v}\sigma$ as arguments. This implies $\overline{v}\sigma$ satisfies condition (iii) of the definition of a fluted sequence. Therefore, $\overline{v}\sigma$ is a fluted sequence. Clearly, the application of a substitution preserves the 'is suffix of' relation, which implies that the argument sequence of $L\sigma$ is a suffix of $\overline{v}\sigma$. Hence, $\overline{v}\sigma$ is the fluted sequence associated with $L\sigma$. Finally then, we can conclude that $(C\backslash\nabla)\sigma$ is a (FL2)-clause. So are $C\sigma$ and $C\sigma \vee D\sigma$.

(c) Both A_1 and A_2 are functional. Suppose $A_1 = P(\overline{u}_1)$ and $A_2 = P(\overline{v}_1)$, and both $(\overline{u}_1)$ and $(\overline{u}_1)$ are m-sequences. We will prove that $(\overline{u}_1)$ is an instance of $(\overline{v}_1)$, or $(\overline{v}_1)$ is an instance of $(\overline{u}_1)$. For then, assuming that $(\overline{u}_1)$ is an instance of $(\overline{v}_1)$, we have that $u_m\sigma = v_m$.[2] Using 3. of Lemma 1, it follows that the fluted sequence $\overline{v}$ associated with $P(\overline{v}_1)$ is a fluted sequence, moreover, it is the fluted sequence for $P(\overline{u}_1)\sigma$.

Our proof that one argument sequence is (a variant of) an instance of the other, is via the consideration of their fluted sequences. Let $\overline{u}$ and $\overline{v}$ denote the fluted sequences of $P(\overline{u}_1)$ and $P(\overline{v}_1)$. Though $(\overline{u}_1)$ and $(\overline{v}_1)$ have equal length, in general, $\overline{u}$ and $\overline{v}$ do not. Suppose the length of $\overline{u}$ is n_1 while the length of $\overline{v}$ is n_2, and $n_1 \geq n_2$. Suppose $\overline{w}$ is the n_2-suffix of $\overline{u}$. Then σ is a most general unifier of $\overline{w}$ and $\overline{v}$ iff it is a most general unifier of $(\overline{u}_1)$ and $(\overline{v}_1)$. To see this note that by Lemma 1.3. $\overline{u}$ is determined by its last argument u_m, which is also the last argument of $(\overline{u}_1)$. Because we know the length of $(\overline{u}_1)$ we can build $(\overline{u}_1)$ from u_m. Similarly, $\overline{w}$ can be built from u_m. Therefore, it suffices to prove that $\overline{w}$ is a variant of an instance of $\overline{v}$, or vice versa.

Suppose $\overline{w} = (w_1, \ldots, w_{n_2})$ and $\overline{v} = (v_1, \ldots, v_{n_2})$, and i is the least index such that either w_i or v_i is a functional term. Suppose v_i is a functional term. (The case that w_i is a functional term, can be proved similarly.) We will iteratively construct a most general

[2] Strictly, $(\overline{u}_1)$ is a variant of the σ-instance of $(\overline{v}_1)$ and hence $u_m\sigma$ is a variant of v_m.

unifier $\sigma' = \theta_1 \dots \theta_n$ of $\overline{w}$ and $\overline{v}$ such that $\overline{w}\sigma' = \overline{v}$. We will show by induction that, for every k where $1 \leq k \leq n_2$, it is possible to construct a binding θ_k such that for every j with $1 \leq j \leq k$ we have that $w_j\theta_1 \dots \theta_j = v_j$. Let $\theta_1 = \{w_1 \mapsto v_1\}, \dots, \theta_{i-1} = \{w_{i-1} \mapsto v_{i-1}\}$. As $w_1, \dots, w_{i-1}$ are distinct variables and $v_1, \dots, v_{i-1}$ are distinct variables, $\theta_1 \dots \theta_{i-1}$ is a variable position preserving renaming, and $w_j\theta_1 \dots \theta_j = v_j$ for each j with $1 \leq j < i$. This deals with the case that both w_j and v_j are variables. Now suppose for some k, $i \leq k \leq n_2$, we have $w_j\theta_1 \dots \theta_j = v_j$ for every j with $1 \leq j < k$. Consider three cases:

(i) Both w_k and v_k are functional terms. By definition, and the fact that w_k and v_k are unifiable, $w_k = f(w_l, \dots, w_{k-1})$ and $v_k = f(v_l, \dots, v_{k-1})$ for some f and $1 \leq l < k$. By the induction hypothesis it follows that $w_k\theta_1 \dots \theta_{k-1} = v_k$. Thus letting θ_k be the identify substitution, we get what we are aiming for: $w_k\theta_1 \dots \theta_{k-1}\theta_k = v_k$.

(ii) w_k is a variable and v_k is a functional term. From the definition of a fluted sequence it follows that w_k is different from all the variables in the domain of $\theta_1 \dots \theta_{k-1}$. This implies $w_k\theta_1 \dots \theta_{k-1} = w_k$ (the occur check does not prevent us to unify w_k and v_k, since w_k cannot occur in v_k). So let $\theta_k = \{w_k \mapsto v_k\}$. Therefore, $w_k\theta_1 \dots \theta_{k-1}\theta_k = w_k\theta_k = v_k$ as required.

(iii) w_k is a functional term and v_k is a variable. As v_i is a functional term all subsequent elements of $\overline{v}$ are functional terms as well. Thus, v_k cannot be a variable.

This completes the proof for (c), and the lemma. □

Lemma 11 Let $C = C' \vee A_1$ and $D = \neg A_2 \vee D'$ be strongly fluted clauses. Suppose A_1 and $\neg A_2$ are eligible literals in C and D, respectively, and suppose σ is the most general unifier of A_1 and A_2. Then $C\sigma \vee D\sigma$ is a strongly fluted clause.

Proof. By considering the different possibilities for C and D. We ignore the propositional case, as either C and D are both propositional or neither are propositional.

1. Both C and D are (FL1)-clauses. Then A_1 and A_2 are variants of an atom $P(\overline{x}_1)$, for some P and $(\overline{x}_1)$. Thus, σ is a variable-position preserving renaming and the result follows.

2. Similarly, if both C and D are strongly fluted (FL3)-clauses, then A_1 and A_2 are variants and σ is a variable-position preserving renaming. Hence $C\sigma$, $D\sigma$ and $C\sigma \vee D\sigma$ are (FL3)-clauses.

3. C is a (FL1)-clause and D is a strongly fluted (FL3)-clause. Then A_2 occurs in the $\mathcal{C}(\overline{x}_1, x_{m+1})$ part of D. It follows that σ is a variable renaming which preserves variable-positions as well as the types of the clauses. Thus, $C\sigma \vee D\sigma$ has the type (FL3).

4. Both C and D are (FL2)-clauses. By Lemma 10, $C\sigma$, $D\sigma$ and $C\sigma \vee D\sigma$ are (FL2)-clauses.

5. C is a (FL1)-clause and D is a (FL2)-clause. This case is a subcase of Lemma 10(b) (with $C \backslash \nabla = \emptyset$). $C\sigma$, $D\sigma$ and $C\sigma \vee D\sigma$ are (FL2)-clauses.

6. C is a strongly fluted (FL3)-clause and D is a (FL2)-clause. The proof is similar as for 5. $C\sigma$, $D\sigma$ and $C\sigma \vee D\sigma$ are (FL2)-clauses.

□

Lemma 12 Let C, D and σ be as in Lemma 11. Then,

$$|\mathrm{var}(C\sigma \vee D\sigma)| \leq \max\{|\mathrm{var}(C)|, |\mathrm{var}(D)|\}.$$

Proof. By inspection of the cases dealt with in the proofs of the previous two lemmas. □

Lemma 13 Removing any subclause from a fluted clause produces a fluted clause.

Proof. For each of the different forms (FL0)–(FL3) deleting a literal produces a clause of the same form as the original clause. □

This cannot be said for strongly fluted clauses, in particular, not for clauses of the form (FL3). For all other forms the statement is also true for strongly fluted clauses, namely, removing any subclause from strongly fluted clauses produces strongly fluted clauses. Consequently:

Lemma 14 The condensation of any (strongly) fluted clause is a (strongly) fluted clause.

Lemma 15 The resolvent of any two strongly fluted clauses is a strongly fluted clause, or, it is only a fluted clause, if one of the premises is a (FL3)-clause.

Lemma 16 Any maximally split, condensed and separated factor or resolvent of strongly fluted clauses is strongly fluted.

Proof. By Lemmas 7, 9, 14 and 15. □

This proves that the class of (strongly) fluted clauses is closed under $\mathsf{R}^{\mathrm{sep}}$-resolution with eager application of condensing, splitting and separation.

In the next three lemmas, N is assumed to be a finite set of fluted clauses (which will be transformed into a set of strongly fluted clauses during the derivation, see Lemma 7). Our goal is to exhibit the existence of a term depth bound of all inferred clauses, as well as the existence of a bound on the number of variables occurring in any inferred clause. The latter follows immediately from Lemmas 6 and 12.

Lemma 17 All clauses occurring in an $\mathsf{R}^{\mathrm{sep}}$-derivation from N contain at most $m + 1$ variables, where m is the maximal arity of any predicate symbol in N.

The definition of fluted clauses places no restriction on the level of nesting of functional terms. But:

Lemma 18 A bound on the maximal term depth of clauses derived by $\mathsf{R}^{\mathrm{sep}}$ from N is m, where m is the maximal arity of any predicate symbol in N.

Proof. The term depth increases only when two (FL2)-clauses are resolved, and one of the eligible literals is functional (cases (b) and (c) in Lemma 10). A consequence of Lemma 10 is that the maximal length of any fluted sequences associated with literals in inferred clauses does not increase. The deepest term in a fluted sequence $\overline{u}$ is the last element u_n, and has maximal depth if the variable prefix consists just of one element. Therefore, if the length of $\overline{u}$ is n the maximal depth of u_n is n. Any argument sequence of a functional fluted clause is the suffix of such a $\overline{u}$, and thus contains u_n. It follows that the maximal term depth for any derived clause is the maximal length of any fluted sequence of a literal in N. This will be m, the maximal arity of any predicate symbol in N. □

Because the signature is extended dynamically during the derivation, it remains to show that separation cannot be performed infinitely often.

Lemma 19 The number of applications of the "Separate"-rule in an $\mathsf{R}^{\mathrm{sep}}$-derivation from N is bounded.

Proof. Define a measure μ on clauses by $\mu(C) = \max\{\mathrm{ar}(L) \mid L \in C\}$, the maximal arity of any literal in C. For any fluted clause $C \vee D$ which is separable into the components C and D, if the newly introduced atom is $q(\overline{x})$ then $\mu(q(\overline{x})) < \mu(C) < \mu(C \vee D)$ and $\mu(q(\overline{x})) < \mu(D) < \mu(C \vee D)$. (Recall, separable fluted clauses have the form (1).) Thus, the measures of the replacement clauses are strictly less than $\mu(C \vee D)$. Because the first criterion of the ordering $\succ$ is the arity of literals, for all other inferences the measure μ does not increase, and the result follows. □

Now, we can state the main theorem of this paper.

Theorem 5 Let φ be any fluted formula and $N = \mathrm{ClsDef}_\Lambda(\varphi)$, where Def_Λ satisfies the restrictions of Lemma 5. Then:

1. Any $\mathsf{R}^{\mathrm{sep}}$-derivation from N (up to redundancy) terminates.
2. φ is unsatisfiable iff the $\mathsf{R}^{\mathrm{sep}}$-saturation (up to redundancy) of N contains the empty clause.

Proof. Termination is a consequence of Lemmas 17–19, while soundness and completeness follows from Theorems 1, 2 and 4, in conjunction with Lemma 19. □

The final theorem gives a rough estimation of an upper bound for the space requirements.

Theorem 6 The number of maximally split, condensed strongly fluted clauses in any $\mathsf{R}^{\mathrm{sep}}$-derivation from N is an $O(m)$-story exponential, where m is the maximal arity of any predicate symbol in N.

Proof. We consider only maximally split and condensed clauses. Given a fixed signature let $\#(p)$ and $\#(f)$ denote the number of available predicate and function symbols, respectively. In a clause of the form (FL1) no two distinct literals have the same predicate symbol and a literal is uniquely identified by its predicate symbol. Hence, the number of possible clauses of the form (FL1) (and ∇) is $3^{\#(p)}$. Similarly, the number of possible clauses of the form (FL3) is bounded by $3^{\#(p)} \cdot 3^{\#(p)}$. Next, we consider functional (FL2)-clauses (with empty ∇). The number of possible functional (FL2)-clauses depends on the number of possible fluted sequences which can be formed. As a fluted sequence is determined by its last element (Lemma 1.3) we need to establish how many different possibilities there are for these. The maximal term depth in the last element of a fluted sequence is m, the maximal arity of any predicate symbol in N (Lemma 18). This implies the number of different last elements in fluted sequences, and thus the number of different fluted sequences, is bounded from above by $\#(f)^m$. By definition, any functional (FL2)-atom is determined by the suffix of a fluted sequence and a predicate symbol. This implies there can be at most $\#(p) \cdot \#(f)^m$ different functional (FL2)-atoms. Therefore, the number of different (FL2)-clauses is bounded from above by $3^{\#(p)} \cdot 3^{\#(p) \cdot \#(f)^m}$. The factor $3^{\#(p)}$ is the bound for the number of possible ∇-subclauses. Thus far we have not considered the fact that new predicate symbols are introduced by the "Separate"-rule. Potentially, one predicate symbol is introduced for every clause of the $3^{2 \cdot \#(p)}$ (FL3)-clauses. Note their arity is at most $m-1$. Therefore the supply of predicate symbols is of the order of $3^{2 \cdot \#(p)}$, with which of the order of $3^{3^{2 \cdot \#(p)} \cdot \#(f)^m}$ number of clauses can be built. In all, the supply of predicate symbols will be an m-story exponential $k = 3^{3^{\cdot^{\cdot^{\cdot^{3^{2 \cdot \#(p)}}}}}}$. It follows that an upper bound for the number of maximally split fluted clauses in any derivation is given by an $O(m)$-story exponential $3^{k \cdot \#(f)^m}$. $\square$

8 Concluding Remarks

Developing a resolution decision procedure for fluted logic turned out to be more complicated than expected. Even though to begin with, clauses are simple in the sense that no nesting of non-nullary function symbols occurs (Lemma 4), the class of fluted clauses is rather complex. It is thus natural to ask whether there is a less complex clausal class which corresponds to fluted logic. The complexity of the class is a result of the ordering we have proposed. This ordering is unusual as it first considers the arity of a literal, while more conventional ordering refinements first consider the depth of a literal. A conventional ordering has the advantage that term depth growth can be avoided completely. This would induce a class of clauses which can be described by these schemes:

$$\text{propositional clauses} \tag{2}$$

$$\mathcal{C}(\overline{x}_1) \vee \mathcal{C}(\overline{x}_2) \vee \ldots \vee \mathcal{C}(x_m) \quad (= \nabla) \tag{3}$$

$$\nabla \vee \mathcal{C}(\overline{x}_1, f(\overline{x}_1)) \vee \mathcal{C}(\overline{x}_2, f(\overline{x}_1)) \vee \ldots \vee \mathcal{C}(x_m, f(\overline{x}_1)) \vee \mathcal{C}(f(\overline{x}_1)) \tag{4}$$

$$\mathcal{C}(\overline{x}_1) \vee \mathcal{C}(\overline{x}_1, x_{m+1}) \vee \mathcal{C}(\overline{x}_2, x_{m+1}) \vee \ldots \vee \mathcal{C}(x_m, x_{m+1}) \vee \mathcal{C}(x_{m+1}) \tag{5}$$

The difference between this class and the class of fluted clauses defined in Section 4 is scheme (4). Clauses satisfying scheme (4) are (FL2)-clauses, but not every (FL2)-clause has the form (4). With separation (on (5)-clauses without an embracing literal) it is possible to stay within the confines of this class. However, the danger with the separation rule is that it could be applied infinitely often. It is open whether there is a clever way of applying the separation rule so that only finitely many new predicate symbols are introduced. For fluted logic with binary converse an example giving rise to an unbounded derivation is the following.

$$\begin{aligned}
&P_2(x_1, x_2, x_3) \vee P_1(f(x_1, x_2, x_3), x_3)\\
&\neg P_2(x_1, x_2, x_3) \vee \neg P_1(x_1, x_2) \vee P_0(x_2, x_3)\\
&\neg Q_1(x_2) \vee \neg P_1(x_1, x_2)
\end{aligned}$$

We do not know whether the following alternative form of the separation rule could help prevent the unbounded introduction of new symbols.

$$\textbf{Separate}'\text{:} \qquad \frac{N \cup \{C \vee D\}}{\begin{array}{ll} N \cup \{ & q_1(x_1, \ldots, x_n) \vee q_2(x_1, \ldots, x_n), \\ & \neg q_1(x_1, \ldots, x_n) \vee C, \\ & \neg q_2(x_1, \ldots, x_n) \vee D\} \end{array}}$$

provided (i) $C \vee D$ is separable such that $\mathrm{var}(C) \not\subseteq \mathrm{var}(D)$ and $\mathrm{var}(D) \not\subseteq \mathrm{var}(C)$, (ii) $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1, \ldots, x_n\}$ for $n \geq 0$, and (iii) q_1, q_2 do not occur in N, C or D.

Noteworthy about fluted logic and the proposed method is that, in order to establish an upper bound on the number of variables in derived clauses, a truly dynamic renaming rule is needed (namely separation). Though renaming is a standard technique for transforming formulae into well-behaved clausal classes, it is usually applied in advance, see for example [7, 15]. From a theoretical point of view whenever it is possible to do the renaming transformations as part of preprocessing, it is sensible to do so. The above example illustrates what could go wrong otherwise. It should be added though that there are instances where renaming on the fly is useful [27]. For fluted logic it is open whether there is a resolution decision procedure which does not require dynamic renaming. Going by the experience with the other solvable classes, for example, Maslov's class $\overline{K}$ [5, 14], where renaming is only necessary when liftable ordering refinements are used, one possibility may be the use of a refinement which is based on a non-liftable ordering. However, it would seem that the problems described in Section 6 are the same with non-liftable orderings. Even if it turns out that there is a resolution decision procedure which does not use separation, one could imagine that the separation rule can have a favourable impact on the performance of a theorem prover; the reasons being that, with separation the size of clauses can be kept small, which is generally desirable, and for fluted logic separation is a cheap operation (Lemma 8).

As noted earlier, the splitting rule is not essential for the results of this paper. The separation rule already facilitates some form of 'weak splitting', because, if C and D are variable disjoint and non-ground subclauses of $C \vee D$ then separation will replace it by $q \vee C$ and $\neg q \vee D$, where q is a new propositional symbol. A closer resemblance to the splitting rule can be achieved by making q minimal in $q \vee C$ and selecting $\neg q$ in $\neg q \vee D$. Nevertheless, splitting has the advantage that more redundancy elimination operations are possible, for example forward subsumption.

The realisation of a practical decision procedure for fluted logic would require a modest extension of one of the many available first-order theorem provers which are based on ordered resolution with an implementation of the separation rule. Modern theorem provers such as SPASS [28, 29] are equipped with a wide range of simplification rules so that reasonable efficiency could be expected.

Acknowledgements

This work has benefitted from discussions (via email) with Bill Purdy. We wish to thank him, and the referees of the short version of this paper [26], for valuable comments.

References

[1] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Logic Computat.*, 4(3):217–247, 1994.

[2] L. Bachmair and H. Ganzinger. Resolution theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 2000. To appear.

[3] L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proc. Third Kurt Gödel Colloquium (KGC'93)*, vol. 713 of *LNCS*, pp. 83–96. Springer, 1993.

[4] H. de Nivelle. A resolution decision procedure for the guarded fragment. In C. Kirchner and H. Kirchner, editors, *Automated Deduction—CADE-15*, vol. 1421 of *LNAI*, pp. 191–204. Springer, 1998.

[5] C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov. *Resolution Method for the Decision Problem*, vol. 679 of *LNCS*. Springer, 1993.

[6] C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 2000. To appear.

[7] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pp. 295–303. IEEE Computer Society Press, 1999.

[8] G. Gargov and S. Passy. A note on Boolean modal logic. In P. P. Petkov, editor, *Mathematical Logic: Proceedings of the 1988 Heyting Summerschool*, pp. 299–309. Plenum Press, 1990.

[9] V. Goranko and S. Passy. Using the universal modality: Gains and questions. *J. Logic Computat.*, 2(1):5–30, 1992.

[10] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[11] A. Herzig. A new decidable fragment of first order logic, 1990. In Abstracts of the Third Logical Biennial, Summer School & Conference in Honour of S. C. Kleene, Varna, Bulgaria.

[12] I. L. Humberstone. Inaccessible worlds. *Notre Dame J. Formal Logic*, 24(3):346–352, 1983.

[13] I. L. Humberstone. The modal logic of 'all and only'. *Notre Dame J. Formal Logic*, 28(2):177–188, 1987.

[14] U. Hustadt and R. A. Schmidt. An empirical analysis of modal theorem provers. *J. Appl. Non-Classical Logics*, 9(4), 1999.

[15] U. Hustadt and R. A. Schmidt. Maslov's class K revisited. In H. Ganzinger, editor, *Automated Deduction—CADE-16*, vol. 1632 of *LNAI*, pp. 172–186. Springer, 1999.

[16] U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, vol. 1761 of *LNAI*, pp. 192–206. Springer, 2000.

[17] W. H. Joyner Jr. Resolution strategies as decision procedures. *J. ACM*, 23(3):398–417, 1976.

[18] H. J. Ohlbach and R. A. Schmidt. Functional translation and second-order frame properties of modal logics. *J. Logic Computat.*, 7(5):581–603, 1997.

[19] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symbolic Computat.*, 2:293–304, 1986.

[20] W. C. Purdy. Decidability of fluted logic with identity. *Notre Dame J. Formal Logic*, 37(1):84–104, 1996.

[21] W. C. Purdy. Fluted formulas and the limits of decidability. *J. Symbolic Logic*, 61(2):608–620, 1996.

[22] W. C. Purdy. Surrogate variables in natural language. To appear in M. Böttner, editor, *Proc. of the Workshop on Variable-Free Semantics*, 1996.

[23] W. C. Purdy. Quine's 'limits of decision'. *J. Symbolic Logic*, 64(4):1439–1466, 1999.

[24] W. V. Quine. Variables explained away. In *Proc. American Philosophy Society*, vol. 104, pp. 343–347, 1960.

[25] R. A. Schmidt. Decidability by resolution for propositional modal logics. *J. Automated Reasoning*, 22(4):379–396, 1999.

[26] R. A. Schmidt and U. Hustadt. A resolution decision procedure for fluted logic. In *Automated Deduction—CADE-17*, Lecture Notes in Artificial Intelligence. Springer, 2000. To appear.

[27] G. S. Tseitin. On the complexity of derivations in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pp. 115–125. Consultants Bureau, New York, 1970.

[28] C. Weidenbach. SPASS, 1999. `http://spass.mpi-sb.mpg.de`.

[29] C. Weidenbach. SPASS: Combining superposition, sorts and splitting. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, Amsterdam, 2000. To appear.