A Resolution Decision Procedure for Fluted Logic

Renate A. Schmidt[1] and Ullrich Hustadt[2]

[1] Department of Computer Science, University of Manchester
Manchester M13 9PL, United Kingdom, schmidt@cs.man.ac.uk
[2] Centre for Agent Research and Development, Manchester Metropolitan University
Manchester M1 5GD, United Kingdom, U.Hustadt@doc.mmu.ac.uk

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 [18], 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 [21, 22, 24]. 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 [19, 26] 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 [19], 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 [25]. 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 [21, 22, 24] 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 [24]. This logic is the most expressive decidable logic in the lattice of fluted logics. In [23] 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 [18] 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. Because of the space limitations all proofs had to be omitted, but can be found in [17].

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, $\varphi \rightarrow \psi$, $\neg\varphi$, $\varphi \land \psi$, 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'.)

$$\forall x_1(\mathsf{cheese\text{-}eater}(x_1) \leftrightarrow \exists x_2(\mathsf{cheese}(x_2) \land \mathsf{eats}(x_1, x_2)))$$

$$\forall x_1(\mathsf{cheese\text{-}lover}(x_1) \leftrightarrow \forall x_2(\mathsf{cheese}(x_2) \rightarrow (\mathsf{eats}(x_1, x_2) \land \mathsf{likes}(x_1, x_2))))$$

$$\forall x_1 x_2(\mathsf{mwmc}(x_1, x_2) \leftrightarrow (\mathsf{married}(x_1, x_2) \land \forall x_3(\mathsf{have\text{-}child}(x_1, x_2, x_3) \rightarrow \exists x_4 \mathsf{married}(x_3, x_4))))$$

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 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

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

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

Split: $$\frac{N \cup \{C \vee D\}}{N \cup \{C\} \mid N \cup \{D\}}$$ 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 $C\sigma$.

Fig. 1. The calculus R

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 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.

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, $\text{var}(C) = X_m$, 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, $\text{var}(C) = X_m$, and for any literal L in C the argument sequence of L is either $(x_i, x_{i+1}, \dots, x_m)$ or $(u_j, u_{j+1}, \dots, u_n)$, where $1 \leq i \leq m$ 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 $1 \leq k \leq m$. $\overline{u}$ 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, $\text{var}(C) = X_{m+1}$, and for any literal L in C, the argument sequence of L is either $(x_1, x_2, \dots, x_m)$ or $(x_i, \dots, x_m, x_{m+1})$, where $1 \leq i \leq m$.

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:

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

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 $\mathrm{var}(u_n) = X_m$.*
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 $(u_j, u_{j+1}, \ldots, u_n)$.*

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.*

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 [20]. For any first-order formula φ, the definitional form obtained by introducing new names for subformulae at positions in Λ will be denoted by $\mathrm{Def}_\Lambda(\varphi)$.

Theorem 2. *Let φ be a first-order formula. For any subset Λ of the set of positions of φ, 1. φ is satisfiable iff $\mathrm{Def}_\Lambda(\varphi)$ is satisfiable, and 2. $\mathrm{Def}_\Lambda(\varphi)$ can be computed in polynomial time.*

In this paper we assume the clausal form of a first-order formula φ, written $\mathrm{Cls}(\varphi)$, 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 $\mathrm{ClsDef}_\Lambda(\varphi)$ is a set of strongly fluted clauses (provided the newly introduced literals have the form $(\neg)Q_\lambda(\overline{x}_i)$).*

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* $\mathrm{ClsDef}_\Lambda(\varphi)$ *is a set of fluted clauses (again, provided the new literals have the form $(\neg)Q_\lambda(\overline{x}_i)$).*

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).

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 $\mathrm{var}(D_1) \not\subseteq$

[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.

$\mathrm{var}(D_2)$ 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 $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1,\dots,x_n\}$ 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 $N \cup \{\neg q(x_1,\dots,x_n) \vee C,\ q(x_1,\dots,x_n) \vee D\}$ is satisfiable.*

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

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

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) $\mathrm{var}(C) \cap \mathrm{var}(D) = \{x_1,\dots,x_n\}$ 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.*

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 $N \cup \{\neg q(x_1,\dots,x_n) \vee C,\ q(x_1,\dots,x_n) \vee D\}$, 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 $N(= N_0), N_1, \dots,$ the clause set $\bigcup_j N_j$ contains the empty clause.*

More generally, this theorem holds also if $\mathsf{R}^{\mathrm{sep}}$ 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 $1 < i \leq m$ 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.*

7 Termination

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

The ordering $\succ$ of $\mathsf{R}^{\mathsf{sep}}$ 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 $\mathsf{R}^{\mathsf{sep}}$ 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 $\mathsf{R}^{\mathsf{sep}}$-inferences.

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

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.

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 $C \vee D$.*

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.*

Lemma 12. *Let C, D and σ be as in Lemma 11. Then,* $|\mathrm{var}(C\sigma \vee D\sigma)| \leq \max\{|\mathrm{var}(C)|, |\mathrm{var}(D)|\}$.

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

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.*

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.*

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.*

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.*

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 an alternative form of the separation rule could help.

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 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 for avoiding dynamic renaming may be by using 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, for, 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] are equipped with a wide range of simplification rules so that reasonable efficiency could be expected.

Acknowledgements. We wish to thank Bill Purdy and the referees 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, eds., *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 *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 *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, eds., *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, ed., *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 *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 *Automated Deduction in Classical and Non-Classical Logics*, vol. 1761 of *LNAI*, pp. 192–206. Springer, 2000.
17. U. Hustadt and R. A. Schmidt. A resolution decision procedure for fluted logic. Technical Report UMCS-00-3-1, University of Manchester, UK, 2000.
18. W. H. Joyner Jr. Resolution strategies as decision procedures. *J. ACM*, 23(3):398–417, 1976.
19. 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.
20. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symbolic Computat.*, 2:293–304, 1986.
21. W. C. Purdy. Decidability of fluted logic with identity. *Notre Dame J. Formal Logic*, 37(1):84–104, 1996.
22. W. C. Purdy. Fluted formulas and the limits of decidability. *J. Symbolic Logic*, 61(2):608–620, 1996.
23. W. C. Purdy. Surrogate variables in natural language. To appear in M. Böttner, ed., *Proc. of the Workshop on Variable-Free Semantics*, 1996.
24. W. C. Purdy. Quine's 'limits of decision'. *J. Symbolic Logic*, 64:1439–1466, 1999.
25. W. V. Quine. Variables explained away. In *Proc. American Philosophy Society*, vol. 104, pp. 343–347, 1960.
26. R. A. Schmidt. Decidability by resolution for propositional modal logics. *J. Automated Reasoning*, 22(4):379–396, 1999.
27. G. S. Tseitin. On the complexity of derivations in propositional calculus. In A. O. Slisenko, ed., *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`.