A Resolution Decision Procedure for the Guarded Fragment with Transitive Guards

Yevgeny Kazakov and Hans de Nivelle

MPI für Informatik, Saarbrücken, Germany
{ykazakov|nivelle}@mpi-sb.mpg.de

Abstract. We show how well-known refinements of ordered resolution, in particular redundancy elimination and ordering constraints in combination with a selection function, can be used to obtain a decision procedure for the guarded fragment with transitive guards. Another contribution of the paper is a special scheme notation, that allows to describe saturation strategies and show their correctness in a concise form.

1 Introduction

The *guarded fragment* $\mathcal{GF}$ of first order logic has been introduced by Andréka, van Benthem & Németi (1998) to explain and generalize the good computational properties of modal and temporal logics. This is achieved essentially by restricting quantifications in first order formulae to the following "bounded" forms: $\forall\overline{x}.[G \to F]$ and $\exists\overline{y}.[G \wedge F]$, where G should be an atomic formula (so-called *guard*) containing all free variables of F. The guarded fragment is decidable in 2EXPTIME (Grädel 1999) and inherits many other nice computational properties from the modal logics like the finite model property, the interpolation property and invariance under an appropriate notion of bisimulation.

Many extensions of the guarded fragment have been found to capture the known formalisms: the *loosely guarded fragment* has been introduced by van Benthem (1997) to capture the until operator in temporal logics; Grädel & Walukiewicz (1999) have extended the guarded fragment by fixed-point constructors to capture the modal mu-calculus. All these extensions of the guarded fragment, however, cannot express the *transitivity axiom:* $\forall xyz.(xTy \wedge yTz \to xTz)$. Transitivity is important, since it is used to model discrete time (in temporal verification) and ordered structures (in program shape analysis). The question of whether transitivity can be safely integrated into the guarded fragment was answered negatively by Grädel (1999). He proved that the guarded fragment becomes undecidable as long as transitivity is allowed. This result was later sharpened by Ganzinger, Meyer & Veanes (1999) who showed that even the *two-variable* guarded fragment $\mathcal{GF}^2$ with transitivity is undecidable. The same paper, however, presents the first restriction of the guarded fragment, where transitivity can be allowed without loss of decidability. In this, so-called *monadic* $\mathcal{GF}^2$, binary relations are allowed to occur as guards only. The paper poses two natural questions: ***(i)*** Does $\mathcal{GF}$ remain decidable if *transitive* predicates are admitted only as guards? ***(ii)*** What is the exact complexity of the monadic $\mathcal{GF}^2$? The first question was answered positively in (Szwast & Tendera 2001) where using a heavy model-theoretic construction, it was shown that the *guarded fragment with transitive guards* $\mathcal{GF}[\mathcal{TG}]$ is decidable in 2EXPTIME. Kieroński (2003) has proved the matching 2EXPTIME lower bound for the *monadic* $\mathcal{GF}^2$ with transitivity, answering hereby the second question.

A practical disadvantage of procedures based on enumeration of structures, like the one given for $\mathcal{GF}[\mathcal{TG}]$ in (Szwast & Tendera 2001), is that without further optimizations, those methods exhibit the full worst-case complexity. Resolution-based approach, is a reasonable alternative to model-theoretic procedures, as its goal-oriented nature and numerous refinements allow to scale well between "easy" and "hard" instances of problems. In this paper we demonstrate the practical power of resolution refinements, such as redundancy elimination and usage of ordering constraints in combination with selection function. We present a first resolution-based decision procedure for $\mathcal{GF}[\mathcal{TG}]$. Another aspect that is demonstrated in our paper is the usage of resolution as a specification language for decision procedures. We introduce a special scheme notation that allows to describe resolution strategies in a concise form. This may provide a formal foundation for using resolution for specifying decision procedures and proving their correctness.

2 Preliminaries

We shall use a standard notation for first-order logic clause logic. An *expression* is either a term or a literal. A *literal symbol* l is either a or $\neg a$, where a is a predicate symbol. An *expression symbol* e is either a functional symbol f or a literal symbol l. We write literals and expressions using literal symbols and expression symbols as follows: $L = l(t_1,...,t_n)$, $E = e(t_1,...,t_n)$. As usual, a clause is a disjunction of literals $C = L_1 \vee \cdots \vee L_n$. The empty clause is denoted by $\Box$. We use the shortcuts $\bowtie$ for conjunction or disjunction and $\overline{x}$ for some vector of variables.

The *depth of an expression* $dp(E)$ is recursively defined as follows: ***(i)*** $dp(x) := 0$; ***(ii)*** $dp(e(t_1,...,t_n)) := \max\{0, dp(t_1),..., dp(t_n)\} + 1$. The *depth of the clause* $C = L_1 \vee \cdots \vee L_n$ is $dp(C) := \max\{0, dp(L_1),..., dp(L_n)\}$. The *width of a formula* $wd(F)$ is the maximal number of free variables in subformulas of F.

2.1 The framework of resolution theorem proving

For describing the decision procedures we use the well-known ordered resolution calculus with selection $\mathcal{OR}^{\succ}_{Sel}$ enhanced with additional simplification rules. Our presentation of the calculus is very close to (Bachmair & Ganzinger 2001). The ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ is parametrized by an admissible ordering $\succ$ and a selection function Sel. A partial ordering $\succ$ on atoms is *admissible* (for $\mathcal{OR}^{\succ}_{Sel}$) if ***(i)*** $\succ$ is *liftable*: $A_1 \succ A_2$ implies $A_1\sigma \succ A_2\sigma$ for any substitution σ and ***(ii)*** $\succ$ is a total reduction ordering on ground atoms. Although resolution remains complete for a much wider class of orderings, admissible orderings are better understood and widely used in existing theorem provers. Examples of admissible orderings are the *recursive path ordering with status* $RPOS$ and the *Knuth-Bendix ordering* KBO.

The ordering $\succ$ is extended on literals by comparing $L = A$ as the multiset $\{A\}$ and $L = \neg A$ as the multiset $\{A, A\}$. The ordering on clauses is the multiset extension of the ordering on literals. Given a clause C, we say that a literal $L \in C$, is *maximal in* C if there is no L' in C, with $L' \succ L$. A *selection function* Sel assigns a set of negative literal to every clause, which we call *selected literals*. A literal L is *eligible* in a clause C if it is either selected: $L \in Sel(C)$, or otherwise nothing is selected and L is maximal in C.

The ordered resolution calculus $\mathcal{OR}^{\succ}_{Sel}$ consists of two inference rules below. We mark eligible literals with "star" and underline the expressions to be unified:

Ordered (Hyper-)Resolution

$$HR\colon \frac{C_1 \vee \underline{A_1}^* \ldots\ C_n \vee \underline{A_n}^* \quad D \vee \neg\underline{B_1}^* \vee \ldots \vee \neg\underline{B_n}^*}{C_1\sigma \vee \ldots \vee C_n\sigma \vee D\sigma} \;\Bigg|\; \textit{where (i) } \sigma = mgu(A_i, B_i)\textit{, (ii) } A_i \textit{ and } \neg B_i \textit{ are eligible } (1 \le i \le n).$$

Ordered Factoring

$$OF\colon \frac{C \vee \underline{A}^* \vee \underline{A'}}{C\sigma \vee A\sigma} \;\Bigg|\; \textit{where (i) } \sigma = mgu(A, A')\textit{, (ii) } A \textit{ is eligible.}$$

The conventional **Ordered Resolution** rule OR, is a partial case of the ordered (hyper-)resolution rule when $n = 1$. The calculus $\mathcal{OR}^{\succ}_{Sel}$ is refutationally complete for any choice of an admissible ordering $\succ$ and a selection function Sel. Moreover, the calculus is compatible with a general notion of redundancy which allows to make use of additional simplification rules.

A ground clause C is called *redundant* w.r.t. a set of the ground clauses N if C follows from the set $N_{\prec C}$ of the clauses from N that are smaller than C. A non-ground clause C is redundant w.r.t. N if every ground instance $C\sigma$ of C is redundant w.r.t. the set N^{gr} of all ground instances of N. A *ground inference* $S \vdash C$ from the clause set S is called *redundant* w.r.t. a clause set N if its conclusion C follows from the set $N^{gr}_{\prec max(S)}$, where $max(S)$ is the maximal clause from S. A non-ground *inference* $S \vdash C$ is redundant w.r.t. N if every ground instance $S\sigma \vdash C\sigma$ of the inference is redundant w.r.t. N. A clause set N is *saturated up to redundancy* if the conclusion of every non-redundant w.r.t. N inference from N is contained in N.

Theorem 1. (Bachmair & Ganzinger 2001) *Let N be a clause set that is saturated up to redundancy in $\mathcal{OR}^{\succ}_{Sel}$. Then N is satisfiable iff N does not contain the empty clause.*

For our decision procedures we do not need the full power of redundancy but rather additional simplification rules. A (non-deterministic) inference rule $S \vdash S_1 \parallel S_2 \cdots \parallel S_k$ producing one of the clause sets S_i from the clause set S is called *sound* if every model of S can be extended to a model for some S_i with $1 \le i \le k$. Additionally, if *every* set S_i makes some clause from S redundant, the rule is called a *simplification* rule.

Given a set of clauses N, a theorem prover based on ordered resolution non-deterministically computes a *saturation* of N by adding conclusions of inference rules to N and marking[1] redundant clauses as *deleted* so that they do not participate in further inferences. If the process terminates without deriving the empty clause $\square$, then a set of the clauses $\mathcal{OR}^{\succ}_{Sel}(N)$ is computed that is saturated in $\mathcal{OR}^{\succ}_{Sel}$ up to redundancy. Theorem 1 then implies that the clause set N is satisfiable, since only satisfiability preserving transformations $N \Rightarrow \cdots \Rightarrow \mathcal{OR}^{\succ}_{Sel}(N)$ were applied to N. Note that termination of a saturation process is a key issue of using resolution as a decision procedure. If any application of inference rules is *a priori* guaranteed to terminate for a clause set N then satisfiability of N can be decided in finite time by enumerating all possible saturations.

In our paper we use the following simplification rule:

Elimination of Duplicate Literals $\quad ED\colon \dfrac{[\![\, C \vee D \vee D \,]\!]}{C \vee D}$

An additional simplification rule will be introduced later, when a certain class of orderings is considered. We indicate redundant premises of rules by enclosing them in double brackets. The simplification rules are applied *eagerly*, that is before any resolution or factoring inference is made. In particular, in the sequel we assume that no clause contain several occurrences of the same literal.

Constraint clauses. The ordered resolution calculus on non-ground level is a directly lifted version of the calculus on the ground level: ***(i)*** each clause represents the set of its ground instances and ***(ii)*** whenever an inference is possible from the ground instances of some clauses, there should be a corresponding inference from the clauses themselves, that captures the result of the ground inference. This is due to the fact that $\mathcal{OR}^{\succ}_{Sel}$ is parametrized with a liftable ordering $\succ$ and does not use non-liftable conditions in inferences (like, say, in the paramodulation calculus, when paramodulation to a variable is not allowed). Therefore, in fact, any representation for sets of ground clauses can be admitted as long as the condition ***(ii)*** above holds. In our decision procedure we use *constraint clauses* of the form: $C \mid R$, where C is a (non-ground) clause and R is a set of *ordering constraints* of the form: $t \succ s$ or $t \succeq s$. Constraint clause $C \mid R$ represent the set of ground instances $C\sigma$ of C such that every constraint in $R\sigma$ is true. The ordered resolution calculus and all notions of redundancy can be straightforwardly adopted to be used with constraint clauses: instead of considering all substitutions (for determining a maximal literal, or showing redundancy) one should consider only substitutions satisfying the constraints. In particular, one could use different values for selection function for different constraint variants of the same clause.

[1] Clauses are not removed from the set to avoid repetition of generation/deletion of the same redundant clauses.

2.2 Schemes of expressions and clauses

To describe resolution-based decision procedures we have to reason about sets of clauses. We introduce a special notation that allows to represent sets of clauses in a compact form. We extend our vocabulary with additional symbols called *signature groups* that represent sets of functional symbols: *function groups*, predicate symbols: *predicate groups* or literal symbols: *literal groups*. We allow to use these symbols in expressions as usual functional and literal symbols and to distinguish them, we use small letters with a "hat" $\hat{g}$. For instance, if $\hat{f}_{all}$ denotes the set of all functional symbols, we write $\hat{f}_{all}(t)$ meaning a term of the form $f(t)$ where $f \in \hat{f}_{all}$ (the formal definition follows below). We adopt the following notation for referring to arguments of expressions. By writing $e\langle !t_1,...,!t_n,s_1,...,s_m\rangle$ we mean an expression starting with the expression symbol e, having all arguments $t_1,...,t_n$ and optional arguments $s_1,...,s_m$ (arranged in some way). Formally, the set of *term schemes*, *literal schemes* and *clause schemes* are defined as follows:

$$\begin{array}{rl}
\hat{T}m ::= & x \mid \hat{f}(\hat{t}_1,...,\hat{t}_n) \mid \hat{f}\langle !\hat{t}_1,...,!\hat{t}_n,\hat{s}_1,...,\hat{s}_m\rangle,\ n\geq 0, m\geq 0.\\
\hat{L}t ::= & \hat{l}(\hat{t}_1,...,\hat{t}_n) \mid \hat{l}\langle !\hat{t}_1,...,!\hat{t}_n,\hat{s}_1,...,\hat{s}_m\rangle,\ n\geq 0, m\geq 0.\\
\hat{C}l ::= & \hat{L} \mid !\hat{L} \mid \hat{C}_1 \vee \hat{C}_2.
\end{array}$$

where $\hat{f}$ is a functional group, $\hat{l}$ is a literal group, $\hat{t}_i$, $\hat{s}_j$ with $1\leq i\leq n$, $1\leq j\leq m$ are term schemes, $\hat{L}$ is a literal scheme and $\hat{C}_1,\hat{C}_2$ are clause schemes. For convenience, we assume that every functional and literal symbol acts as a singleton group consisting of itself, so usual terms and clauses are term schemes and clause schemes as well.

Each term scheme $\hat{t}$, literal scheme $\hat{L}$ and clause scheme $\hat{C}$ represents a set $\langle\hat{t}\rangle$, $\langle\hat{L}\rangle$ and $\langle\hat{C}\rangle$ of terms, literals and clauses respectively, as defined below:

$$\begin{array}{rrl}
\langle\hat{T}m\rangle,\langle\hat{L}t\rangle := & \langle x\rangle\colon & \{x\} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid\\
& \langle\hat{g}(\hat{t}_1,...,\hat{t}_n)\rangle\colon & \{g(t_1,...,t_n) \mid g\in\hat{g},\ t_i\in\langle\hat{t}_i\rangle, 1\leq i\leq n\} \qquad \mid\\
\multicolumn{2}{r}{\langle\hat{g}\langle !\hat{t}_1,...,!\hat{t}_n,\hat{s}_1,...,\hat{s}_m\rangle\rangle\colon} & \{g(h_1,...,h_k) \mid g\in\hat{g},\ \{h_1,...,h_k\}\cap\langle\hat{t}_i\rangle\neq\emptyset, 1\leq i\leq n,\\
& & \qquad \{h_1,...,h_k\}\subseteq\cup_{i=1}^{n}\langle\hat{t}_i\rangle\cup_{j=1}^{m}\langle\hat{s}_j\rangle\}.
\end{array}$$

$$\begin{array}{rrl}
\langle\hat{C}l\rangle = & \langle\hat{L}\rangle\colon & \{L_1\vee\cdots\vee L_k \mid \boldsymbol{k\geq 0},\ L_i\in\langle\hat{L}\rangle,\ 1\leq i\leq k\} \quad \mid\\
& \langle !\hat{L}\rangle\colon & \{L_1\vee\cdots\vee L_k \mid \boldsymbol{k\geq 1},\ L_i\in\langle\hat{L}\rangle,\ 1\leq i\leq k\} \quad \mid\\
& \langle\hat{C}_1\vee\hat{C}_2\rangle\colon & \{C_1\vee C_2 \mid C_1\in\langle\hat{C}_1\rangle,\ C_2\in\langle\hat{C}_2\rangle\}.
\end{array}$$

We use the shortcuts $\hat{e}(.,\overline{x},.)$, $\hat{e}\langle .,\overline{x},.\rangle$ and $\hat{e}\langle .,!\overline{x},.\rangle$ where $\overline{x}$ is a vector $x_1,...,x_n$, to stand for $\hat{e}(.,x_1,...,x_n,.)$, $\hat{e}\langle .,x_1,...,x_n,.\rangle$ and $\hat{e}\langle .,!x_1,...,!x_n,.\rangle$ respectively. We write $..\vee\neg!\hat{A}\vee..$ in clause schemes instead of $..\vee !\neg\hat{A}\vee..$, where $\hat{A}$ is either of the form $\hat{a}(...)$ or $\hat{a}\langle...\rangle$. In fact we use variable vectors $\overline{x}$ and functional symbols without "hat" f as *parameters of clause schemes*. A clause scheme $\hat{C}(\overline{x},f,...)$ with parameters $\overline{x},f,...$ represents the union $\langle\hat{C}\rangle := \cup_\eta\langle C\eta\rangle$ for all substitutions η of vectors $x_1,...,x_n$ for $\overline{x}$, function symbols for f, etc..

Example 1. Suppose $\hat{a}$ is a predicate group consisting of all predicate symbols and $\hat{\alpha} := \{\hat{a},\neg\hat{a}\}$ is a literal group consisting of all literal symbols. Then the clause scheme $\hat{C} = \neg !\hat{a}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle ! f(\overline{x}), \overline{x}\rangle$ has two parameters: $\overline{x}$ and f. Any clause $C \in \langle\hat{C}\rangle$ corresponds to some choice of these parameters $\overline{x} = x_1, ..., x_n$, $f = f'$. The clause C should have a nonempty subset of negative literals containing all variables $x_1, ..., x_n$ and no other arguments. Other literals of C should contain the subterm $f'(x_1, ..., x_n)$ as an argument and possibly some variables from $x_1, ..., x_n$. In particular, $\langle\hat{C}\rangle$ contains the clauses $\neg a(x, y, x) \vee b(y, f'(x, y))$, $\neg b(x, y) \vee \neg b(y, x)$ and $\neg p \vee \neg q(c, c)$, but not the clauses $\neg a(x, y, x) \vee b(f'(x, y), f'(y, x))$ or $\neg b(y, f'(x, y))$.

3 Deciding the guarded fragment by resolution

In this section we demonstrate our technique by revisiting a resolution decision procedure for the guarded fragment without equality. The original procedure is due to (de Nivelle & de Rijke 2003). Resolution-based decision procedures (for an overview see Fermüller, Leitsch, Hustadt & Tammet 2001) usually consist of several main steps. First, a clause normal form transformation is applied to a formula of a fragment that produce *initial clauses*. Then a clause set containing the initial clauses is defined, that is shown later to be closed under inferences of the calculus. Decidability and complexity results follow from the fact that the defined clause class contains only finitely many different clauses over a fixed signature.

3.1 Clause normal form translation

In order to describe the transformation to a *clause normal form* (CNF), it is convenient to use the *recursive definition* for the guarded fragment:

$$\mathcal{GF} ::= \mathtt{A} \mid \mathtt{F}_1 \vee \mathtt{F}_2 \mid \mathtt{F}_1 \wedge \mathtt{F}_2 \mid \neg\mathtt{F}_1 \mid \forall\overline{x}.(\mathtt{G} \rightarrow \mathtt{F}_1) \mid \exists\overline{x}.(\mathtt{G} \wedge \mathtt{F}_1).$$

where $\mathtt{A}$ is an atom, $\mathtt{F}_i, i = 1, 2$ are guarded formulas, and $\mathtt{G}$ is an atom called *the guard* containing all free variables of $\mathtt{F}_1$. The translation of a guarded formula into CNF is done in two steps. First, the formula is transformed into *negation normal form* (NNF) in the standard way. Guarded formulas in NNF are defined by the following recursive definition:

$$[\mathcal{GF}]^{nnf} ::= (\neg)\mathtt{A} \mid \mathtt{F}_1 \vee \mathtt{F}_2 \mid \mathtt{F}_1 \wedge \mathtt{F}_2 \mid \forall\overline{y}.(\mathtt{G} \rightarrow \mathtt{F}_1) \mid \exists\overline{y}.(\mathtt{G} \wedge \mathtt{F}_1).$$

Second, a so-called *structural transformation* is applied, that decomposes the formula by introducing *definitions* for all of its subformulae. We assume that to each subformula $\mathtt{F}'$ of $\mathtt{F}$, a unique predicate $P_{\mathtt{F}'} = p_{\mathtt{F}'}(\overline{x})$ is assigned. Each predicate $P_{\mathtt{F}'}$ has the arity equal to the number of free variables $\overline{x}$ of $\mathtt{F}'$. Using the new predicates, the structural transformation can be defined as $\exists\overline{x}.P_{\mathtt{F}} \vee [\mathtt{F}]^{st}$, where $[\mathtt{F}]^{st}$ is given below. In each row, $\overline{x}$ are the free variables of $\mathtt{F}$.

$$\begin{array}{rcl}
[\mathtt{F}]_g^{st} := [(\neg)\mathtt{A}]_g^{st} : \forall\overline{x}.(P_{\mathtt{F}} \rightarrow (\neg)\mathtt{A}) & | & \neg p_{\mathtt{F}}(\overline{x}) \vee (\neg)a\langle\overline{x}\rangle \\
[\mathtt{F}_1 {\vee\!\!\!\!\wedge} \mathtt{F}_2]_g^{st} : \forall\overline{x}.(P_{\mathtt{F}} \rightarrow [P_{\mathtt{F}_1} {\vee\!\!\!\!\wedge} P_{\mathtt{F}_2}]) \wedge [\mathtt{F}_1]_g^{st} \wedge [\mathtt{F}_2]_g^{st} & | & \neg p_{\mathtt{F}}(\overline{x}) \vee p_{\mathtt{F}_i}\langle\overline{x}\rangle \ [\vee\ p_{\mathtt{F}_j}\langle\overline{x}\rangle] \\
[\forall\overline{y}.(\mathtt{G} \rightarrow \mathtt{F}_1)]_g^{st} : \forall\overline{x}.(P_{\mathtt{F}} \rightarrow \forall\overline{y}.[\mathtt{G} \rightarrow P_{\mathtt{F}_1}]) \wedge [\mathtt{F}_1]_g^{st} & | & \neg g\langle !\overline{x}, !\overline{y}\rangle \vee \neg p_{\mathtt{F}}(\overline{x}) \vee p_{\mathtt{F}_1}\langle\overline{x}, \overline{y}\rangle \\
[\exists y.\mathtt{F}_1]_g^{st} : \forall\overline{x}.(P_{\mathtt{F}} \rightarrow \exists y.P_{\mathtt{F}_1}) \wedge [\mathtt{F}_1]_g^{st}. & & \neg p_{\mathtt{F}}(\overline{x}) \vee p_{\mathtt{F}_1}\langle f(\overline{x}), !\overline{x}\rangle
\end{array}$$

The transformation unfolds a guarded formula according to its construction and introduces predicates and definitions for its guarded subformulae. A guarded formula $\mathtt{F}$ in negation normal form is satisfiable whenever $\exists\overline{x}.P_{\mathtt{F}} \wedge [\mathtt{F}]^{st}$ is: one can extend the model of $\mathtt{F}$ by interpreting the new predicate symbols according to their definitions. Every recursive call of the transformation contributes to a result with a conjunct describing a definition for an introduced predicate. Performing the usual skolemization and writing the result in a clause form, we obtain the clauses shown to the right of the definition for $[\mathtt{F}]_g^{st}$. It is easy to see that the clauses for $P_{\mathtt{F}} \wedge [\mathtt{F}]_g^{st}$ fall into the set of clauses described by the following clause schemes:

$$\begin{array}{l} 1.\ \hat{\alpha}\langle\hat{c}\rangle; \\ 2.\ \neg!\hat{a}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle. \end{array} \tag{G}$$

where the predicate group $\hat{a}$ consists of all (initial and introduced) predicate symbols and the literal group $\hat{\alpha}$ consists of all literal symbols.

3.2 Saturation of the clause set

The resolution calculus has two parameters that can be chosen: an admissible ordering and a selection function. These parameters should prevent clauses from growing during the inferences. We will set the ordering and selection function in such a way, that eligible literals would be ***(i)*** of maximal depth and ***(ii)*** contain all variables of the clause.

We assume that the ordering $\succ$ enjoys $L \succ K$ for $L \in \langle\hat{a}\langle !f(\overline{x}),\overline{x}\rangle\rangle$ and $K \in \langle\hat{a}\langle\overline{x}\rangle\rangle$, that is, any literal containing the functional symbol with all variables is greater then any other literal in the clause without functional subterms. This can be achieved by taking, say, any recursive path ordering $\succ_{rpos}$ on expressions with the precedence $>_P$ enjoying $f >_P p$ for any functional symbol f and predicate symbol p. We define the selection function Sel for the clauses without functional symbols to select a negative literal containing all variables of the clause if there is one.

We prove that the clause class from (G) is closed under the ordered resolution by making case analysis of possible inferences between clauses of this class. The complete case analysis is given below:

1 $\hat{\alpha}^*$	2 $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle$
1.1 $\hat{\alpha}\langle\hat{c}\rangle \vee \underline{\hat{a}\langle\hat{c}\rangle}^*$: OR.1	2.1 $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \hat{\alpha}\langle !f(\overline{x}),\overline{x}\rangle^*$
1.2 $\hat{\alpha}\langle\hat{c}\rangle \vee \underline{\neg\hat{a}\langle\hat{c}\rangle}^*$: OR.2	2.1.1 $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \underline{\hat{a}\langle !f(\overline{x}),\overline{x}\rangle}^*$: OR.1
1.3 $\hat{\alpha}\langle\hat{c}\rangle \vee \underline{\hat{a}\langle\hat{c}\rangle}^* \vee \underline{\hat{a}\langle\hat{c}\rangle}$: OF	2.1.2 $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \underline{\neg\hat{a}\langle !f(\overline{x}),\overline{x}\rangle}^*$: OR.2
OR[1.1; 1.2]: $\hat{\alpha}\langle\hat{c}\rangle$: 1	2.1.3 $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \underline{\hat{a}\langle !f(\overline{x}),\overline{x}\rangle}^* \vee \underline{\hat{a}\langle f(\overline{x}),\overline{x}\rangle}$: OF
OF[1.3] : $\hat{\alpha}\langle\hat{c}\rangle \vee \hat{a}\langle\hat{c}\rangle$: 1	OR[2.1.1; 2.1.2]: $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle$: 2
	OF[2.1.3] : $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \hat{a}\langle !f(\overline{x}),\overline{x}\rangle$: 2
	2.2 $\neg\underline{\hat{g}\langle !\overline{x}\rangle}^* \vee \neg\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle\overline{x}\rangle$: OR.2
	OR[1.1; 2.2] : $\hat{\alpha}$: 1
	OR[2.1.1; 2.2]: $\neg!\hat{g}\langle !\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle \vee \hat{\alpha}\langle f(\overline{x}),\overline{x}\rangle$: 2

The table is organized as follows. The clause schemes from (G) are spread in the table on different levels of precision. On the first level the schemes are given themselves. On the second level, different possibilities for eligible literals (marked by the asterisk) are considered. On the last level, possible inference rules that can be applied for a clause are identified and the expressions to be unified are underlined. For example, OR.1 marked to the right of the clause scheme 1.1 means that a clause represented by this scheme may act as a first premise of the ordered resolution rule. Below the last level, inferences between preceding clauses are drawn and their conclusions are identified as instances of clause schemes.

We have used the special form of literals in the clauses when the unifiers has been computed. For instance, the reason of why the resolution inference OR[2.1.1; 2.1.2] has produced the clause of the same depth is because the so-called *covering* expressions have been unified. An expression E is called *covering* if all functional subterms of E contain *all variables* of E. It is well known that the unifier for the two covering expressions is the renaming for the deepest of them:

Theorem 2. (Fermüller, Leitsch, Tammet & Zamov 1993) *Let E_1 and E_2 be two covering expressions with $dp(E_1) \geq dp(E_2)$ and let $\sigma = mgu(E_1, E_2)$. If $\overline{x} = free(E_1)$ then $\sigma : \overline{x} \to \overline{u}$ for some vector of variables $\overline{u}$. As a conclusion $dp(E_2\sigma) = dp(E_1\sigma) = dp(E_1)$.*

Theorem 3. (de Nivelle & de Rijke 2003) *Ordered resolution decides the guarded fragment in double exponential time.*

Proof. Given a formula $F \in \mathcal{GF}$ of size n, the structural transformation introduces at most linear number of new predicate symbols of arity not greater than n. Since every non-ground clause from (G) has a guard, the number of variables in such a clause does not exceed n. It can be shown that most $c = 2^{2^{O(n \log n)}}$ different clauses from (G) over the initial and introduced signature can be constructed. A saturation of the size c can be computed in time $O(c^2)$. So the resolution decision procedure for $\mathcal{GF}$ can be implemented in 2EXPTIME. $\square$

4 Deciding the guarded fragment with transitivity

Some binary predicates of Σ, which we call *transitive predicates* have a *special* status. We usually denote them by the letters T, S and use the *infix* notation $(t_1 T t_2)$ rather than the *postfix* notation $a(t_1, t_2)$, as for the other predicates. For any group of transitive predicates $\hat{T} = \{T_1, \ldots, T_n\}$, the shortcuts $(x\hat{T}y)$ and $\neg(x\hat{T}y)$ represent respectively the disjunctions $(xT_1y) \vee \cdots \vee (xT_ny)$ and $\neg(xT_1y) \vee \cdots \vee \neg(xT_ny)$. We assume that every set of clauses N contains the *transitivity clause*: $\neg(xTy) \vee \neg(yTz) \vee xTz$ for every transitive predicate T.

The *guarded fragment with transitive guards* $\mathcal{GF}[\mathcal{TG}]$ is defined by:

$$\mathcal{GF}[\mathcal{TG}] ::= \mathtt{A} \mid \mathtt{F}_1 \vee \mathtt{F}_2 \mid \mathtt{F}_1 \wedge \mathtt{F}_2 \mid \neg\mathtt{F}_1 \mid \forall\overline{x}.(\mathtt{G} \to \mathtt{F}_1) \mid \exists\overline{x}.(\mathtt{G} \wedge \mathtt{F}_1).$$

where $\mathtt{F}_i, i = 1, 2$ are from $\mathcal{GF}[\mathcal{TG}]$, $\mathtt{A}$ is a non-transitive atom and $\mathtt{G}$ is a (possibly transitive) guard for $\mathtt{F}_1$. Note that $\mathcal{GF}$ can be seen as a sub-fragment of

$\mathcal{GF}[\mathcal{TG}]$, when there are no transitive predicates. It is easy to see from the CNF transformation for guarded formulas that transitive predicates can appear only in initial clauses of the form: $\neg xTy \vee \hat{\alpha}\langle x, y\rangle$, $\neg xTx \vee \hat{\alpha}\langle x\rangle$ or $\neg g(x) \vee T\langle x, f(x)\rangle$. We present a resolution decision procedure for $\mathcal{GF}[\mathcal{TG}]$ as an extension of the one for $\mathcal{GF}$ by carefully analyzing and blocking the cases when resolution with transitivity predicates can lead to unbounded generation of clauses.

4.1 Obstacles for deciding the guarded fragment with transitivity

The transitivity clauses do not behave well when they resolve with each other because the number of variables increases. The simple solution is to block the inferences between the transitivity axioms by setting the selection function *Sel* such that it selects one of the negative literals. However this is only a partial solution to the problem since saturation with other clauses, in which positive transitive literals "should" be maximal, generate arbitrary large clauses as shown on the example below (left part):

1. $\neg(\underline{xTy})^* \vee \neg(yTz) \vee xTz$;	1. $\neg(\underline{xTy})^* \vee \neg(\underline{yTz})^* \vee xTz$;
2. $\alpha(x) \vee \underline{f(x)Tx}^*$;	2. $\alpha(x) \vee \underline{f(x)Tx}^*$;
$\mathtt{OR}[2;1]$: 3. $\alpha(x) \vee \neg(xTz) \vee \underline{f(x)Tz}^*$;	$\mathtt{HR}[2,2;1]$: 3. $\alpha(x) \vee \underline{ff(x)Tx}^*$;
$\mathtt{OR}[3;1]$: 4. $\alpha(x) \vee \neg(xTz) \vee \neg(zTz_1) \vee \underline{f(x)Tz_1}^*$;	$\mathtt{HR}[3,2;1]$: 4. $\alpha(x) \vee \underline{fff(x)Tx}^*$;
......:	.........: ....

The reason for the growth of the clause size is that the atoms which were resolved in the inferences *do not contain all variables* of the clause. To keep the number of variables from growing it is possible to use the *hyperresolution*, namely to select both negative literals of the transitivity clause and resolve them simultaneously. However, this strategy may result in increase of the clause depth, as shown on the right part of the example. Note that the variable depth in hyperresolution inference with the transitivity clause grows only if for the terms h, t and s which where simultaneously unified with x, y and z respectively, either $h \succ max(t, s)$ or $s \succ max(t, h)$. In all other cases, say, when $h = t \succ s$ like in the inference below, neither variable depth nor the number of variables grows:

1. $\neg(\underline{xTy})^* \vee \neg(\underline{yTz})^* \vee xTz$; 2. $\alpha(x) \vee \underline{xTx}^*$; 3. $\beta(x) \vee \underline{f(x)Tx}^*$;

$\mathtt{HR}[2,3;1]$: 4. $\alpha(f(x)) \vee \beta(x) \vee f(x)Tx$;

We are going to distinguish these cases of using the transitivity clauses by using *ordering constraints* in combination with a selection function. We split the transitivity clause into the *constraint* clauses of forms:

$$
\begin{array}{lll}
\mathtt{T} & \neg(xTy) \vee \neg(yTz) \vee xTz; & \\
\hline
\mathtt{T.1.} & \neg(xTy)^* \vee \neg(yTz) \vee xTz & \mid x \succ max(y, z); \\
\mathtt{T.2.} & \neg(xTy) \vee \neg(yTz)^* \vee xTz & \mid z \succ max(y, x); \\
\mathtt{T.3.} & \neg(xTy)^* \vee \neg(yTx)^* \vee xTx & \mid x \succ y; \\
\mathtt{T.4.} & \neg(xTy)^* \vee \neg(yTz)^* \vee xTz & \mid y \succeq max(x, z);
\end{array}
\qquad (\mathrm{T})
$$

where selected literals are indicated with the asterisk. In the sequel, assume that every set of clauses contains transitivity clauses T.1 – T.4 from (T) for every transitive predicate T.

4.2 Redundancy of inferences involving transitive relations

In this section we prove the main technical lemmas that allow to gain a control over the saturation process in presence of transitivity clauses. We show that many inferences involving transitive predicates are redundant. It is not very convenient to show redundancy of inferences "by definition". We proof auxiliary lemmas using which redundancy of inferences can be shown in a much simpler way.

Lemma 1 (Four Clauses). *Let N be a clause set containing the ground clauses:*
$\texttt{C1}.\ C \vee C' \vee \underline{A}^*;\quad \texttt{C2}.\ D \vee D' \vee \neg\underline{A}^*;\quad \texttt{C3}.\ C \vee D \vee B;\quad \texttt{C4}.\ C' \vee D' \vee \neg B;$
Then the following ordered resolution inference:
$\texttt{OR}[\texttt{C1};\texttt{C2}]{:}\texttt{P}.\quad C \vee C' \vee D \vee D';$ *is redundant provided that $A \succ B$.*

Proof. Obviously, the conclusion of the inference $\texttt{OR}[\texttt{C1};\texttt{C2}]$ follows from the clauses C3 and C4. It remains to show that both C3 and C4 are smaller than the maximum of the clauses C1 and C2. We use the fact that the conclusion of the ordered resolution inference is always smaller than the premise with the negative eligible literal. Therefore, $C \vee D \prec \texttt{P} \prec \texttt{C2}$ and since $B \prec \neg A \prec \texttt{C2}$, $\texttt{C3} = C \vee D \vee B \prec \texttt{C2}$. Similarly, $\texttt{C4} \prec \texttt{C2}$. We have shown that $max(\texttt{C3},\texttt{C4}) \prec max(\texttt{C1},\texttt{C2})$, thus the inference $\texttt{OR}[\texttt{C1};\texttt{C2}]$ is redundant. □

Lemma 1 can be generalized to show redundancy of hyperresolution inferences as follows:

Lemma 2. *Let N be a clause set containing the ground clauses:*
$\texttt{C1}.\ C_1 \vee \underline{A_1}^*;\ \cdots\ \texttt{Cn}.\ C_n \vee \underline{A_n}^*;\qquad \texttt{D1}.\ C'_1 \vee D'_1;\ \cdots\ \texttt{Dm}.\ C'_m \vee D'_m;$
$\texttt{C}.\ \ C \vee \neg\underline{A_1}^* \vee \cdots \vee \neg\underline{A_n}^*;$
*for $n, m > 1$ such that: **(i)** $C_1 \vee \cdots \vee C_n \vee C = C'_1 \vee \cdots \vee C'_m$, **(ii)** $D'_1 \wedge \cdots \wedge D'_m \models \bot$ and **(iii)** $max(A_1, \ldots, A_n) \succ max(D'_1, \ldots, D'_m)$. Then the (hyper-)resolution inference:* $\texttt{HR}[\texttt{C1},\texttt{C2},\ldots,\texttt{Cn};\texttt{C}]{:}\texttt{P}.\quad C_1 \vee C_2 \vee \cdots \vee C_n \vee C;$ *is redundant.*

Proof. The conclusion $C_1 \vee \cdots \vee C_n \vee C = C'_1 \vee \cdots \vee C'_n$ of the inference logically follows from the clauses $\texttt{D1},\ldots,\texttt{Dm}$ because of the condition (ii). Moreover, $max(\texttt{D1},\ldots,\texttt{Dm}) \prec max(\texttt{C1},\ldots,\texttt{Cn},\texttt{C}) = \texttt{C}$ since for any i with $1 \le i \le m$, $C'_i \prec \texttt{P} \prec \texttt{C}$ (condition (i)) and $D'_i \prec \neg A_1 \vee \cdots \vee \neg A_n$ (condition (iii)). Therefore, the inference $\texttt{HR}[\texttt{C1},\texttt{C2},\ldots,\texttt{Cn};\texttt{C}]$ is redundant. □

For proving redundancy of inferences involving transitive relations, we need to make additional assumption about the the ordering $\succ$ used in $\mathcal{OR}^{\succ}_{Sel}$. We say that the ordering $\succ$ is *T-argument monotone* if: **(i)** $\{t_1, t_2\} \succ_{mul} \{s_1, s_2\}$ implies $(t_1 T t_2) \succ (s_1 T s_2)$, and **(ii)** $b(t_1, t_2) \succ (t_1 T t_2) \succ u(t_1)$ for any non-transitive predicate b and unary predicate u. From now on we assume that the ordering $\succ$ is T-argument monotone. The intended ordering can be easily obtained from the ordering $\succ_{rpos}$, that has been used for deciding the guarded fragment, by requiring that all transitive predicates have the multiset status and $b >_P T >_P u$ for any non-transitive predicate b whose arity is greater than two, transitive predicate T and unary predicate u.

Lemma 3. *Let N be a clause set containing the clause:*
1. $C \vee \underline{t_1 T t_2}^*$; *together with the result of the inference:*
(a) $\mathtt{OR}[1;\mathtt{T.1}]$*:* 2. $C \vee \neg(t_2 T z) \vee \underline{t_1 T z}^* \mid t_1 \succ max(t_2, z)$; *or*
(b) $\mathtt{OR}[1;\mathtt{T.2}]$*:* 2. $C \vee \neg(x T t_1) \vee \underline{x T t_2}^* \mid t_2 \succ max(t_1, x)$;
Then the following inferences are redundant respectively:
(a) $\mathtt{OR}[2;\mathtt{T.1}]$*:* $C \vee \neg(t_2 T z) \vee \neg(z T z_1) \vee t_1 T z_1 \mid t_1 \succ max(t_2, z, z_1)$;
(b) $\mathtt{OR}[2;\mathtt{T.2}]$*:* $C \vee \neg(x_1 T x) \vee \neg(x T t_1) \vee x_1 T t_2 \mid t_2 \succ max(t_1, x, x_1)$.

Proof. **(a)** The result of any instance of the inference $\mathtt{OR}[2;\mathtt{T.1}]$:
$2a.$ $C \vee \neg(t_2 T s) \vee \underline{t_1 T s}^* \mid t_1 \succ max(t_2, s)$;
$\mathtt{T.1}a.$ $\neg(\underline{t_1 T s})^* \vee \neg(s T h) \vee t_1 T h \mid t_1 \succ max(s, h)$;
$\mathtt{OR}[2a;\mathtt{T.1}a]$: $C \vee \neg(t_2 T s) \vee \neg(s T h) \vee t_1 T h \mid t_1 \succ max(t_2, s, h)$;
can be obtained from other instances of the constraint clauses 2 and $\mathtt{T}$:
$2b.$ $C \vee \neg(\underline{t_2 T h}) \vee t_1 T h \mid t_1 \succ max(t_2, h)$;
$\mathtt{T}b.$ $\neg(t_2 T s) \vee \neg(s T h) \vee \underline{t_2 T h}$;
by resolving on the smaller atom: $t_2 T h \prec t_1 T s$. Therefore, by Lemma 1 the inference is redundant. The case (b) is proven symmetrically to (a). □

Lemma 4. *Let N be a clause set containing the clauses:*
1. $C \vee \underline{t_1 T t_2}^*$;
2. $D \vee \underline{t_2 T t_3}^*$;
$\mathtt{OR}[1;\mathtt{T.2}]$ *:* 3. $C \vee \neg(x T t_1) \vee \underline{x T t_2}^* \mid t_2 \succ max(t_1, x)$;
$\mathtt{OR}[2;\mathtt{T.1}]$ *:* 4. $D \vee \neg(t_3 T z) \vee \underline{t_2 T z}^* \mid t_2 \succ max(t_3, z)$;
$\mathtt{HR}[1,2;\mathtt{T.4}]$*:* 5. $C \vee D \vee t_1 T t_3 \mid t_2 \succeq max(t_1, t_3)$;
$\mathtt{HR}[2,3;\mathtt{T.3}]$*:* 6. $D \vee C \vee \neg(t_3 T t_1) \vee t_2 T t_2 \mid t_2 \succ t_3$;
Then the following inferences are redundant:
(a) $\mathtt{HR}[1,4;\mathtt{T.4}]$*:* $C \vee D \vee \neg(t_3 T z) \vee t_1 T z \mid t_2 \succ max(t_3, z);\ t_2 \succeq t_1$
(b) $\mathtt{HR}[3,2;\mathtt{T.4}]$*:* $C \vee D \vee \neg(x T t_1) \vee x T t_3 \mid t_2 \succ max(t_1, x);\ t_2 \succeq t_3$
(c) $\mathtt{HR}[3,4;\mathtt{T.4}]$*:* $C \vee D \vee \neg(x T t_1) \vee \neg(t_3 T z) \vee x T z \mid t_2 \succ max(t_1, t_3, x, z)$;
(d) $\mathtt{HR}[4,3;\mathtt{T.3}]$*:* $D \vee C \vee \neg(t_3 T x) \vee \neg(x T t_1) \vee t_2 T t_2 \mid t_2 \succ max(t_1, t_3, x)$.

Proof. The proof is analogous to Lemma 4. The complete proof can be found in the extended version of the paper (de Nivelle & Kazakov 2004). □

We have shown that redundancy and ordering constraints help to avoid many inferences involving transitivity. However, certain inferences may still result in increasing the number of variables in clauses as in the situation shown below:
1. $\alpha(x) \vee \underline{f(x) T x}^*$;
2. $\neg(\underline{x T y})^* \vee a(x) \vee \beta(y)$;
3. $\neg(\underline{x T y})^* \vee \neg a(x) \vee \beta'(y)$;
$\mathtt{OR}[1;\mathtt{T.1}]$: 5. $\alpha(x) \vee \neg(x T z) \vee \underline{f(x) T z}^* \mid f(x) \succeq max(x, z)$;
$\mathtt{OR}[5;2]$: 6. $\alpha(x) \vee \neg(x T z) \vee \underline{a(f(x))}^* \vee \beta(z) \mid f(x) \succeq max(x, z)$;
$\mathtt{OR}[5;3]$: 7. $\alpha(x) \vee \neg(x T z_1) \vee \underline{\neg a(f(x))}^* \vee \beta'(z_1) \mid f(x) \succeq max(x, z_1)$;
$\mathtt{OR}[6;7]$: 8. $\alpha(x) \vee \alpha(x) \vee \neg(x T z) \vee \neg(x T z_1) \vee \beta(z) \vee \beta'(z_1) \mid f(x) \succeq max(x, z, z_1)$;
.... : ....

The problem here is that the functional term $f(x)$ which does not contain all variables of the clause appears as an argument of a non-transitive predicate. That has happened as result of resolution inferences $\mathtt{OR}[5;2]$ and $\mathtt{OR}[5;3]$. To resolve this problem we introduce an additional inference rule:

Transitive Recursion

$$TR\colon \frac{\neg(x\hat{T}y)^{*} \vee \alpha(x) \vee \beta(y)}{\begin{array}{c}\neg(x\hat{T}y) \vee \alpha(x) \vee u^{\hat{T}}_{\alpha(\cdot)}(y)\\ \neg(x\hat{T}y) \vee \neg u^{\hat{T}}_{\alpha(\cdot)}(x) \vee u^{\hat{T}}_{\alpha(\cdot)}(y)\\ \neg u^{\hat{T}}_{\alpha(\cdot)}(y) \vee \beta(y)\end{array}}$$

where (i) $\hat{T}$ is a not empty set of transitive predicates (ii) $u^{\hat{T}}_{\alpha}$ is a special unary predicate indexed by α and $\hat{T}$.

The inference rule extends the signature by introducing new unary predicate symbols $u^{\hat{T}}_{\alpha}$, whose intended interpretation is "the set of elements that are T-reachable from the ones where α is false".

Lemma 5. *The transitive recursion rule is a sound inference rule.*

Proof. Let $\mathcal{M}$ be a model for the premise of the rule, such that all predicates $T_1, \ldots, T_n$ from $\hat{T}$ are interpreted by transitive relations and let $xT'y := xT_1y \wedge \cdots \wedge xT_ny$. Obviously, T' is interpreted in $\mathcal{M}$ by a transitive relation. We extend $\mathcal{M}$ to a the model $\mathcal{M}'$ by interpreting the new predicate $u^{\hat{T}}_{\alpha}(x)$ as the formula $\exists x'.(\neg\alpha(x') \wedge x'T'x)$. In particular, $\mathcal{M}' \models \forall y.([\exists x.(\neg\alpha(x) \wedge xT'y)] \rightarrow u^{\hat{T}}_{\alpha}(y))$, so the first conclusion of the inference rule is true in $\mathcal{M}'$. The following sequence of implications: $u^{\hat{T}}_{\alpha}(x) \wedge xT'y \;\equiv\; \exists x'.[\neg\alpha(x') \wedge x'T'x \wedge xT'y] \Rightarrow$ (transitivity of T') $\Rightarrow\; \exists x'.[\neg\alpha(x') \wedge x'T'y] \;\equiv\; u^{\hat{T}}_{\alpha}(y)$ shows that the second conclusion is true in $\mathcal{M}'$. Finally, the last conclusion is a consequence of the premise of the rule: $u^{\hat{T}}_{\alpha}(y) \;\equiv\; \exists x'.(\neg\alpha(x') \wedge x'T'y) \;\Rightarrow\; \exists x'.\beta(y) \;\equiv\; \beta(y)$. □

Note that the transitive recursion rule is not a reduction rule, although the premise logically follows from the conclusion (the premise may be smaller than a clause in the conclusion, say, when $\beta(y)$ is empty). However, the rule helps to avoid dangerous inferences involving transitive predicates, like in the example above, by making them redundant:

Lemma 6. *Let $\hat{T} = \{T_1, \ldots, T_n\}$ with $n \geq 1$ be the set of transitive predicates and N be a clause set containing the clauses:*

$\mathtt{D}.\ \neg(\underline{x\hat{T}z})^{*} \vee \alpha(x) \vee \beta(z);$	$1^{i}.\ C_i \vee \underline{tT_ih}^{*}$ *with one of the following clauses:*
$\mathtt{D}_1.\ \neg(\underline{x\hat{T}z})^{*} \vee \alpha(x) \vee u(z);$	$\mathtt{OR}[1^{i};\mathtt{T.1}]\colon 2^{i}_{a}.\ C_i \vee \neg(hT_iz) \vee \underline{tT_iz}^{*} \mid t \succ max(h,z);$
$\mathtt{D}_2.\ \neg(\underline{x\hat{T}z})^{*} \vee \neg u(x) \vee u(z);$	$\mathtt{OR}[1^{i};\mathtt{T.2}]\colon 2^{i}_{b}.\ C_i \vee \neg(xT_it) \vee \underline{xT_ih}^{*} \mid h \succ max(t,x);$
$\mathtt{D}_3.\ \neg u(z) \vee \beta(z);$	$\mathtt{HR}[1^{1}, \ldots, 1^{n}; D_1]\colon 3.\ C_1 \Cup C_n \vee \alpha(t) \vee u(h).$

for $1 \leq i \leq n$. Then the following inferences are redundant respectively:

(a) $\mathtt{HR}[2^{1}_{a}, \ldots, 2^{n}_{a}; \mathtt{D}]\colon\ \Cup_{i=1}^{n}\{C_i \vee \neg hT_iz\} \vee \alpha(t) \vee \beta(z)$
(b) $\mathtt{HR}[2^{1}_{b}, \ldots, 2^{n}_{b}; \mathtt{D}]\colon\ \Cup_{i=1}^{n}\{C_i \vee \neg xT_it\} \vee \alpha(x) \vee \beta(h)$

Proof. Consider the case (a) (case (b) is proven symmetrically). For any instance of the inference $\mathtt{HR}[2^{1}_{a}, \ldots, 2^{n}_{a}; \mathtt{D}]$ satisfying the constraints:

2_a^i. $C_i \vee \neg(hT_is) \vee \underline{tT_is}^* \mid t \succ max(h,s)$;
D. $\neg(\underline{t\hat{T}s})^* \vee \alpha(t) \vee \beta(s)$;
HR[$2_a^1, \ldots, 2_a^n$; D]: $\mathbb{W}_{i=1}^n\{C_i \vee \neg hT_is\} \vee \alpha(t) \vee \beta(s)$

the conclusion can be derived from the clause 3 and instances of D_2 and D_3:

3. $C_1 \mathbb{W} C_n \vee \alpha(t) \vee \underline{u(h)}$.
$\mathtt{D}_2^a$. $\neg(h\hat{T}s) \vee \neg\underline{u(h)} \vee \underline{u(s)}$;
$\mathtt{D}_3^a$. $\neg\underline{u(s)} \vee \beta(s)$;

by resolving on $u(h)$ and $u(s)$, both of which are smaller than each tT_is used in the inference. Therefore the inference is redundant by Lemma 2. □

Remark 1. Note that the inferences HR[$2_a^1, \ldots, 2_a^n$; $\mathtt{D}_1$] and HR[$2_a^1, \ldots, 2_a^n$; $\mathtt{D}_2$] (HR[$2_b^1, \ldots, 2_b^n$; $\mathtt{D}_1$] and HR[$2_b^1, \ldots, 2_b^n$; $\mathtt{D}_2$]) are redundant as well, since we can apply Lemma 6 for $\beta(z) := u(z)$; and $\alpha(x) := \neg u(x)$, $\beta(z) := u(z)$ respectively.

4.3 Saturation of the clause set

We have prepared the ground for describing a resolution decision procedure for $\mathcal{GF}[\mathcal{TG}]$. However, to simplify the upcoming case analysis, we introduce an additional inference rule:

Literal Projection $LP: \dfrac{[\![\, C \vee L \,]\!]}{\begin{array}{c} p_L(x) \vee C \\ \neg p_L(x) \vee L \end{array}}$ *where (i) L is non-unary literal with* free$[L] = \{x\}$; *(ii) C contains x in non-unary literal or in functional subterm and (iii) p_L is a unary predicate for L.*

The literal projection rule is a variant of the general splitting rule, which allows to split a clause by introducing a *new* predicate over shared variables of its parts. The purpose of this rule is to avoid clauses with several positive transitive literals that can be produced, for instance, with the inference:

1. $\neg\underline{a(f(x))}^* \vee xT_1x$; 2. $\neg b(x) \vee \underline{a(f(x))}^* \vee xT_2x$;
OR[1; 2]: $\neg b(x) \vee xT_1x \vee xT_2x$;

Instead of producing the inference above, one can alternatively simplify the clauses 1 and 2 using the literal projection rule:

1*a*. $p_{T_1}(x) \vee \neg\underline{a(f(x))}^*$ 2*a*. $p_{T_2}(x) \vee \neg b(x) \vee \underline{a(f(x))}^*$;
1*b*. $\neg p_{T_1}(x) \vee \underline{xT_1x}$; 2*b*. $\neg p_{T_2}(x) \vee xT_2x$;
OR[1*a*; 2*a*]: $p_{T_1}(x) \vee p_{T_2}(x) \vee \neg b(x)$;

Note that the literal projection rule cannot be applied to literals containing a new predicate symbol since they are unary, therefore, only finitely many predicates p_L can be introduced.

We show the decidability of $\mathcal{GF}[\mathcal{TG}]$ in similar way as for $\mathcal{GF}$ by describing a clause class containing the input clauses for $\mathcal{GF}[\mathcal{TG}]$-formulae and closed under the ordered resolution inferences up to redundancy. This clause class is represented by the set of the clause schemes below:

$$
\begin{array}{ll}
1: & [\neg\hat{T},\hat{\gamma}]\langle\hat{c}\rangle; \\
2: & [\neg!\hat{d},\hat{\gamma}]\langle!\overline{x}\rangle \vee [\neg\hat{T},\hat{\gamma}]\langle!f(\overline{x}),\overline{x}\rangle \vee \hat{\beta}\langle f(\overline{x}),\overline{x}\rangle; \\
3: & \neg!\hat{p}^1(!x) \vee \neg\hat{p}^1(f(x)) \vee \hat{\mathcal{T}}\langle f(x),x\rangle; \\
\mathtt{T}: & \neg(xTy) \vee \neg(yTz) \vee xTz; \\
4: & \neg!\hat{p}^1(x) \vee \neg(xTz) \vee f(x)Tz \mid f(x) \succ z; \\
5: & \neg!\hat{p}^1(x) \vee \neg(zTx) \vee zTf(x) \mid f(x) \succ z; \\
\mathtt{R}: & \neg(x!\hat{T}y) \vee \hat{\gamma}(x) \vee \hat{\gamma}(y);
\end{array}
\qquad
\begin{array}{l}
\hat{\alpha} := \{\hat{a},\neg\hat{a}\}; \\
\hat{\mathcal{T}} := \{\hat{T},\neg\hat{T}\}; \\
\hat{b} := \{\hat{a},p_{\hat{\alpha}},p_{\hat{T}(\cdot)}\}; \\
\hat{\beta} := \{\hat{b},\neg\hat{b}\}; \\
\hat{p} := \{\hat{b},u^{\mathcal{T}}_{\beta}\}; \hat{\gamma} := \{\hat{p},\neg\hat{p}\}; \\
\hat{d} := \{\hat{p},\hat{T}\};\ \hat{\delta} := \{\hat{d},\neg\hat{d}\}.
\end{array}
\qquad \text{(GT)}
$$

Where $\hat{a}$ consists of initial non-transitive predicate symbols and $\hat{T}$ consists of all transitive predicate symbols.

Theorem 4. *There is a strategy based on $\mathcal{OR}^{\succ}_{Sel}$ with ordering constraints and additional inference rules such that given a formula $F \in \mathcal{GF}[\mathcal{TG}]$ a finite clause set N containing the CNF transformation for F is produced such that:* ***(i)*** *N is closed under rules of $\mathcal{OR}^{\succ}_{Sel}$ up to redundancy and* ***(ii)*** *N is a subset of (GT).*

Proof. The limited space does not allow us to present the complete case analysis of possible inferences between the clauses of (GT). The proof can be found in (de Nivelle & Kazakov 2004), where the extended version of the paper is given. □

Corollary 1. (Szwast & Tendera 2001) *$\mathcal{GF}[\mathcal{TG}]$ is decidable in double exponential time.*

Proof. Given a formula $F \in \mathcal{GF}[\mathcal{TG}]$, it could be seen from construction of (GT) that clauses generated in the saturation for F contain at most linear number of initial predicate and functional symbols and at most exponential number of introduced (by inferences extending the signature) *unary* predicates. Simple calculations show that the number of clauses from (GT) that can be constructed from them is at most double exponential. Therefore the saturation can be computed in double exponential time. □

5 Conclusions and future work

The resolution decision procedure for $\mathcal{GF}[\mathcal{TG}]$ presented in the paper can shed light on the reasons why this fragment is so fragile with respect to decidability and which decidable extensions it may have. Note, that we in fact have already shown the decidability of a larger fragment: it is possible to admit non-empty *conjunctions* of transitive relations $x\hat{T}y$ as guards since the CNF-transformation maps them to the same decidable fragment. This might help to find a decidable counterpart for the *interval-based* temporal logics à-la Halpern Shoham (Halpern & Shoham 1986) because the relation between intervals can be expressed as a conjunction of (transitive) relations between their endpoints. As a future work, we try to extend our approach to the case with equality, as well as to other theories like theories of general compositional axioms: $\forall xyz.(xSy \wedge yTz \rightarrow xHz)$ and theories of linear, branching and dense partial orderings without endpoints.

References

Andréka, H., van Benthem, J. & Németi, I. (1998), 'Modal languages and bounded fragments of predicate logic', *Journal of Philosophical Logic* **27**, 217–274.

Bachmair, L. & Ganzinger, H. (2001), Resolution theorem proving, *in* A. Robinson & A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

de Nivelle, H. & de Rijke, M. (2003), 'Deciding the guarded fragments by resolution', *Journal of Symbolic Computation* **35**, 21–58.

de Nivelle, H. & Kazakov, Y. (2004), Resolution decision procedures for the guarded fragment with transitive guards, Research Report MPI-I-2004-2-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany.

Fermüller, C., Leitsch, A., Hustadt, U. & Tammet, T. (2001), Resolution decision procedures, *in* A. Robinson & A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 25, pp. 1791–1849.

Fermüller, C., Leitsch, A., Tammet, T. & Zamov, N. (1993), *Resolution Methods for the Decision Problem*, Vol. 679 of *LNAI*, Springer, Berlin, Heidelberg.

Ganzinger, H., Meyer, C. & Veanes, M. (1999), The two-variable guarded fragment with transitive relations, *in* 'Proc. 14th IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, pp. 24–34.

Grädel, E. (1999), 'On the restraining power of guards', *Journal of Symbolic Logic* **64(4)**, 1719–1742.

Grädel, E. & Walukiewicz, I. (1999), Guarded fixed point logic, *in* 'Proceedings of 14th IEEE Symposium on Logic in Computer Science LICS '99, Trento', pp. 45–54.

Halpern, J. Y. & Shoham, Y. (1986), A propositional modal logic of time intervals, *in* 'Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986', IEEE Computer Society Press, Washington, DC, pp. 279–292.

Kieroński, E. (2003), The two-variable guarded fragment with transitive guards is 2EXPTIME-hard, *in* A. D. Gordon, ed., 'FoSSaCS', Vol. 2620 of *Lecture Notes in Computer Science*, Springer, pp. 299–312.

Szwast, W. & Tendera, L. (2001), On the decision problem for the guarded fragment with transitivity, *in* 'Proc. 16th IEEE Symposium on Logic in Computer Science', pp. 147–156.

van Benthem, J. (1997), Dynamic bits and pieces, Technical Report LP-97-01, ILLC, University of Amsterdam.