Deciding the First-Order Theory of an Algebra of Feature Trees with Updates (Extended Version)

Nicolas Jeannerod, Ralf Treinen

Deciding the First-Order Theory of an Algebra of Feature Trees with Updates (Extended Version)*

Nicolas Jeannerod and Ralf Treinen

Univ. Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243, CNRS, Paris, France
nicolas.jeannerod@irif.fr ralf.treinen@irif.fr

Abstract. We investigate a logic of an algebra of trees including the update operation, which expresses that a tree is obtained from an input tree by replacing a particular direct subtree of the input tree, while leaving the rest unchanged. This operation improves on the expressivity of existing logics of tree algebras, in our case of feature trees. These allow for an unbounded number of children of a node in a tree.
We show that the first-order theory of this algebra is decidable *via* a weak quantifier elimination procedure which is allowed to swap existential quantifiers for universal quantifiers. This study is motivated by the logical modeling of transformations on UNIX file system trees expressed in a simple programming language.

1 Introduction

Feature trees are trees where nodes have an unbounded number of children, and where edges from nodes to their children carry names such that no node has two different outgoing edges with the same name. Hence, the names on the edges can be used to select the different children of a node. Feature trees have been used in constraint-based formalisms in the field of computational linguistics (e.g. [14]) and constrained logic programming [1,15]. This work is motivated by a different application of feature trees: they are a quite accurate model of UNIX file system trees. The most important abstraction in viewing a file structure as a tree is that we ignore multiple hard links to files. Our mid-term goal is to derive, using symbolic execution techniques, from a shell script a logical formula that describes the semantics of this script as a relation between the initial file tree and the one that results from execution of the script.

Feature tree logics have at their core basic constraints like $x[f]y$, expressing that y is a subtree of x accessible from the root of x via feature f, and $x[f] \uparrow$, expressing that the tree x does not have a feature f at its root node. This is already sufficient to describe some tree languages that are useful in our context. For instance, the script consisting of the single command `mkdir /home/john`,

* This work has been partially supported by the ANR project CoLiS, contract number ANR-15-CE25-0001.

which creates a directory `john` under the directory `home`, succeeds on a tree if the tree satisfies the formula $\exists d.(r[\texttt{home}]d \wedge d[\texttt{john}]\uparrow)$, which expresses that `home` is a subdirectory of the root, which does itself *not* have a subdirectory `john`. We ignore here the difference between directories and regular files, as well as file permissions.

Update Constraints. In order to describe the effect of executing the above script we need more expressivity. A first idea is to introduce an update constraint $y \doteq x[f \mapsto z]$, which states that the tree y is obtained from the tree x by setting its child f to z, and creating the child when it does not exist. Using this, the semantics of `mkdir /home/john` could be described by

$$\exists d, d', e.\, (in[\texttt{home}]d \wedge d[\texttt{john}]\uparrow \wedge out \doteq in[\texttt{home} \mapsto d'] \wedge d' \doteq d[\texttt{john} \mapsto e] \wedge e[\emptyset])$$

Here, $e[\emptyset]$ expresses that e is an empty directory. Note that this formula, by virtue of the update constraint, expresses that any existing directories under `home` different from `john` are not touched.

Programming constructs translate to combinations of logical formula. For instance, if $\phi_p(in, out)$, resp. $\phi_q(in, out)$ describe the semantics of script fragments p and q, then their composition is described by $\exists t.(\phi_p(in, t) \wedge \phi_q(t, out))$. The reality of our use case is more complex than that due to the hairy details of error handling in shell scripts [10], and is up to future work.

Formulas with more complex quantification structure occur when we express interesting properties of scripts. For instance, p and q are equivalent if

$$\forall in, out.\, (\phi_p(in, out) \leftrightarrow \phi_q(in, out))$$

Debian requires in its policy [7] so-called maintainer scripts to be idempotent, which can be expressed for a script p as

$$\forall in, out.\, (\phi_p(in, out) \leftrightarrow \exists t.(\phi_p(in, t) \wedge \phi_p(t, out))$$

Since we are interested in verifying these kinds of properties on scripts we need a logic of feature trees including update constraints, and which enjoys a decidable first-order logic.

Related Work. The first decidability result of a full first-order theory of *Herbrand trees* (*i.e.*, based on equations $x = f(x_1, \ldots, x_n)$) is due to Malc'ev [13], this result has later been extended by [6, 12]. A first decidability result for the first-order theory of *feature trees* was given for the logic FT [1], which comprises the predicates $x[f]y$ and $x[f]\uparrow$, by [4]. This was later extended to the logic CFT [15], which in addition to FT has an *arity constraint* $x[F]$ for any finite set F of feature symbols, expressing that the root of x has precisely the features F, in [3,5]. Note that in these logics one can only quantify over trees, not over feature symbols. The generalization to a two-sorted logic which allows for quantification over features is undecidable [16], but decidability can be recovered if one restricts the use of feature variables to talk about existence of features only [17]. All these decidable logics of trees have a non-elementary lower bound [18]. The case of a feature logic with update constraints was open up to now.

Choosing the Right Predicates. The difficulty in solving update constraints stems from the fact that an update constraint involves three trees: the original tree, the final tree and the sub-tree that gets grafted on the original tree.

There are no symmetries between these three arguments, and a conjunction of several update constraints may become quite involved. Our approach to handle this rather complex update constraint is to replace it by a more elementary constraint system which is based on the classical $x[f]y$, and the new *similarity constraint* $x \sim_f y$. The latter constraint expresses that x and y have the same children with the same names, except for the name f where they may differ. This system has the same expressive power as update constraints since on the one hand $z \doteq x[f \mapsto y]$ is equivalent to $x \sim_f z \wedge z[f]y$, and on the other hand $x \sim_f y$ is equivalent to $\exists z, v.(z \doteq x[f \mapsto v] \wedge z \doteq y[f \mapsto v])$. In order to simplify these constraints one needs the generalization $x \sim_F y$ where F is a finite set of features. For each set of features F, similarities $\sim_F$ are equivalence relations, which is very useful when designing simplification rules, and these relations have useful properties, like $(x \sim_F y \wedge x \sim_G y) \leftrightarrow x \sim_{F \cap G} y$ and $(x \sim_F y \wedge y \sim_G z) \rightarrow x \sim_{F \cup G} z$.

Eliminating Quantifiers. Our theory of feature trees does not have the property of quantifier elimination in the strict sense [9]. This is already the case without the update (or similarity) constraints, as we can see in the following example: $\exists x.(y[f]x \wedge x[g]\uparrow)$. This formula means that there is a tree denoted by x such that y points to x through the feature f, and that x does not have the feature g. A quantifier elimination procedure would have to conserve this information about the global variable y. This situation is not unusual when designing decision procedures. There are basically two possible remedies: the first one is to extend the logical language by new predicates which express properties which otherwise would need existential quantifiers to express. This approach of achieving the property of quantifier elimination by extension of the logical language is well-known from Presburger arithmetic, it was also used in [3, 4].

However, in the case of feature tree logics, the needed extension of the language is substantial and requires the introduction of *path constraints*. For instance, the above formula would be equivalent to the path constraint $y[f][g]\uparrow$ stating that the variable y has a feature f pointing towards a tree where there is no feature g. Unfortunately, this extension entails the need of quite complex simplification rules for these new predicates.

The alternative solution is to our knowledge due to [13] and consists in exploiting the fact that certain predicates of the logic behave like functions. This solution was also used in [6] for Herbrand trees. When switching to feature trees this solution becomes quite elegant [17], the above formula would be replaced by $\neg y[f]\uparrow \wedge \forall x.(y[f]x \rightarrow x[g]\uparrow)$ stating that y has a feature f (by $\neg y[f]\uparrow$) and that for each variable x such that y points towards x via f (in fact, there is only one), x has no feature g. The price is that existential quantifiers are not completely eliminated but swapped for universal ones. This is, however, sufficient, since one can now apply this transformation to a formula in prenex normal form, and successively reduce the number of quantifier eliminations.

Structure of this Paper. We summarize some notions from logic that will be used in the rest of the paper in Section 2. Our model of trees as well as the syntax and semantics of our logic are defined formally in Section 3. The quantifier elimination procedure in given in Section 4. We conclude in Section 5. Proofs are only sketched, full proofs are to be found in the appendixes A and B.

This is the extended version of [11].

2 Preliminaries

We assume logical conjunction and disjunction to be associative and commutative, and equality to be symmetric. For instance, we identify the formula $x \doteq y \wedge (x[f]\uparrow \vee x[g]z)$ with $(x[g]z \vee x[f]\uparrow) \wedge y \doteq x$.

The set of free variables of a formula ϕ is written $\mathcal{V}(\phi)$. We write $\phi\{x \mapsto y\}$ for the formula obtained by replacing in ϕ all free occurrences of x by y. We write $\tilde{\exists}\phi$ for the existential closure $\exists\mathcal{V}(\phi).\phi$, and similarly $\tilde{\forall}\phi$ for $\forall\mathcal{V}(\phi).\phi$.

A *conjunctive clause with existential quantifiers*, or in short *clause*, is either $\bot$, or a finite set of literals prefixed by a string of existential quantifiers. Note that such a clause may still contain free variables, that is we do *not* require all its variables to be quantified. If $\exists X.(a_1 \wedge \ldots \wedge a_n)$ is such a clause, then we can partition its set of literals $c = g_c \cup l_c$ such that g_c contains all the literals of c that contain no variable of X, and l_c the set of literals of c that contain at least one variable of X. We have the following logical equivalence:

$$\models (\exists X.c) \leftrightarrow (g_c \wedge \exists X.l_c)$$

We call (g_c, l_c) the *decomposition* of $\exists X.c$. g_c is the *global part* and l_c the *local part* of c, X is the set of *local variables* and $\mathcal{V}(\exists X.c) \supseteq \mathcal{V}(g_c)$ the set of *global variables*.

A *disjunctive normal form* (dnf) is a finite set of clauses, all of which are different from $\bot$.

A formula is in *prenex normal form* (pnf) if it is of the form $Q_1x_1 \ldots Q_nx_n.\phi$ where ϕ is quantifier-free, and where the Q_i are existential or universal quantifiers. If all Q_i are $\exists$ (resp. $\forall$) then the formula is called a Σ_1-formula (resp. Π_1-formula).

$A \rightsquigarrow B$ denotes the set of partial functions from the set A to the set B with a finite domain. The domain of a partial function f is written $\mathtt{dom}(f)$. The complement of a set is written X^c. We write $X \setminus Y$ for $\{x \in X \mid x \notin Y\}$.

3 A Logic for an Algebra of Trees with Similarities

3.1 Decorations

In addition to what has been said in the introduction, our model of feature trees also has information attached to the *nodes* of the trees. In our application to UNIX filesystems, these could be records containing the usual file attributes like

various timestamps and access permission bits, owner and group, and so on. This work abstracts from the details of the information attached to tree nodes: we take the definition of node decorations, and the pertaining logic as a parameter. We hence assume given an arbitrary set $\mathcal{D}$ of *decorations*.

We assume given a set D of predicate symbols for decorations, and an interpretation $\mathcal{D}$ for D with universe $\mathcal{D}$. These predicate symbols are of course assumed to be disjoint from the predicate symbols that will be introduced in Subsection 3.3. We also require that D contains a binary predicate $x \not\cong y$ expressing the *disequality* of two information items.

We also assume that we have a quantifier elimination procedure for $\mathcal{D}$: we can compute for any Σ_1 formula ψ over the language D, possibly with free variables, a quantifier-free formula $D\text{-elim}(\psi)$ that is equivalent in $\mathcal{D}$ to ψ and has the same free variables. Furthermore, we can decide for any closed and quantifier-free D-formula whether in holds in $\mathcal{D}$.

3.2 Feature Trees

We assume given an infinite set $\mathcal{F}$ of *features*. The letters f, g, h will always denote features.

The set $\mathcal{FT}$ of *feature trees* is inductively defined as

$$\mathcal{FT} = \mathcal{D} \times (\mathcal{F} \rightsquigarrow \mathcal{FT})$$

Here, the case of a partial function with empty domain serves as base case of the induction. Hence, this amounts to saying that a feature tree is a finite unordered tree where nodes are labeled by decorations, and edges are labeled by features. Each node in a feature tree has a finite number of outgoing edges, and all outgoing edges of a node carry different names. We write $\hat{t}$ for the decoration of the root node of t and we write $\vec{t}$ for its mapping at the root, i.e. $t = (\hat{t}, \vec{t}\,)$. Our notion of equality on trees is *structural equality*, i.e. $t = s$ iff $\hat{t} = \hat{s}$ and $\vec{t} = \vec{s}$, that is $\mathtt{dom}(\vec{t}\,) = \mathtt{dom}(\vec{s}\,)$ and $\vec{t}\,(f) = \vec{s}\,(f)$ for every $f \in \mathtt{dom}(\vec{t}\,)$. Examples of feature trees are given in Figure 1.

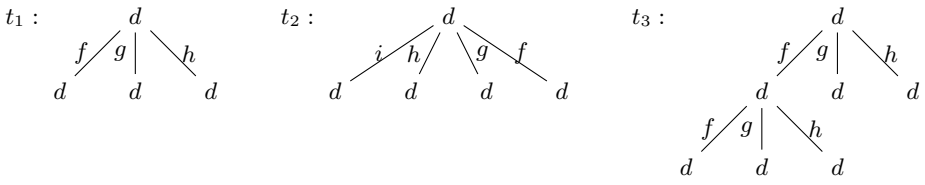

Fig. 1. Examples of Feature Trees. $d \in \mathcal{D}$ is some arbitrary node decoration.

For the reasons explained in the introduction, our logical language does not contain $y \doteq z[x \mapsto f]$ but the simpler $x \sim_F y$ for any *finite* set $F \subseteq \mathcal{F}$. If $F \subseteq \mathcal{F}$ then we say that t *is similar to* s *outside* F, written $t \sim_F s$, if for all $f \in F^c = \mathcal{F} \setminus F$ we have that

– either $f \notin \mathtt{dom}(\vec{t}\,) \cup \mathtt{dom}(\vec{s}\,)$
– or $f \in \mathtt{dom}(\vec{t}\,) \cap \mathtt{dom}(\vec{s}\,)$, and $\vec{t}\,(f) = \vec{s}\,(f)$.

In other words, t and s are similar outside F if they have precisely the same children except maybe for the features in F.

3.3 Constraints and their Interpretation

The set of predicate symbols (or atomic constraints) of our logic is

$x \doteq y$	Equality	$A(x_1 \dots x_n)$	Decoration predicate $A \in D$
$x[f]y$	Feature f from x to y	$x[f]\uparrow$	Absence of feature f from x
$x[F]$	Fence constraint	$x \sim_F y$	Similarity outside F

In fences and similarities, the sets F are finite. We will use the usual syntactic sugar and write $x \not\doteq y$ for $\neg(x \doteq y)$, and $x \not\sim_F y$ for $\neg(x \sim_F y)$. As with equality, we consider similarity predicates to be symmetric, that is we identify $x \sim_F y$ with $y \sim_F x$.

We have one model which has as universe the set $\mathcal{FT}$. As usual, we use the same symbol $\mathcal{FT}$ for the model and for its universe. The predicate symbols are interpreted as follows, where ρ is a valuation of the free variables of the formula in the model $\mathcal{FT}$:

$$
\begin{array}{ll}
\mathcal{FT}, \rho \models x \doteq y & \text{iff } \rho(x) = \rho(y) \\
\mathcal{FT}, \rho \models x[f]y & \text{iff } f \in \mathtt{dom}(\overrightarrow{\rho(x)}) \text{ and } \overrightarrow{\rho(x)}(f) = \rho(y) \\
\mathcal{FT}, \rho \models x[f]\uparrow & \text{iff } f \notin \mathtt{dom}(\overrightarrow{\rho(x)}) \\
\mathcal{FT}, \rho \models x[F] & \text{iff } \mathtt{dom}(\vec{x}\,) \subseteq F \\
\mathcal{FT}, \rho \models x \sim_F y & \text{iff } \rho(x) \sim_F \rho(y) \\
\mathcal{FT}, \rho \models A(x_1, \dots, x_n) & \text{iff } \mathcal{D}, (\lambda x_i.\widehat{\rho(x_i)}) \models A(x_1, \dots, x_n)
\end{array}
$$

Example 1. Let ρ be the valuation $[x \to t_1, y \to t_2, z \to t_3]$ for the trees defined in Figure 1. The following formulas are satisfied in $\mathcal{FT}, \rho$:

$$z[f]x, \quad x[i]\uparrow, \quad x[\{f,g,h,i\}], \quad x \sim_{\{i\}} y$$

Similarity constraints are actually only of interest in case of an infinite set of features. In case of a finite set $\mathcal{F}$, the similarity constraint could already be expressed in the logic FT that was mentioned in Section 1:

$$x \sim_G y \Leftrightarrow \bigwedge_{f \in \mathcal{F} \setminus G} ((x[f]\uparrow \wedge y[f]\uparrow) \vee \exists z(x[f]z \wedge y[f]z))$$

Note the difference between our *fence* constraint, which states an upper bound on the root features of a tree, and the *arity* constraint of [3, 15] which states a precise set of root features of a tree. Both are equivalent, since one can express a fence F as a disjunction of all the arities that are subsets of F. Reciprocally, in our logic, we can express that x has arity F as $x[F] \wedge \bigwedge_{f \in F} \neg x[f]\uparrow$.

Also note that decoration predicates behave in $\mathcal{FT}$ as in $\mathcal{D}$:

Proposition 1. *If ψ is a formula using only symbols of D then*

$$\mathcal{FT}, \alpha \models \psi \qquad \Leftrightarrow \qquad \mathcal{D}, \lambda x.\widehat{\alpha(x)} \models \psi$$

4 Quantifier Elimination

4.1 Clashing Clauses

We say that a clause c that is not $\bot$ *clashes* if one of the patterns of Figure 2 matches (modulo associativity and commutativity of $\wedge$) a sub-clause $c' \subseteq c$.

C-Cycle	$x_1[f_1]x_2 \wedge \ldots \wedge x_n[f_n]x_1$	$(n \geq 1)$
C-Feat-Abs	$x[f]y \wedge x[f]\uparrow$	
C-Feat-Fen	$x[f]y \wedge x[F]$	$(f \notin F)$
C-Neq-Refl	$x \not\doteq x$	
C-NSim-Refl	$x \not\sim_F x$	

Fig. 2. Clash patterns

Remark that C-Cycle is a clash since our model allows for finite feature trees only, the other clash cases should be obvious.

Lemma 1. *If a clause c clashes then $\mathcal{FT} \models (c \to \bot)$.*

4.2 Positive Clauses with Local Variables

As a preparation for the general case we first consider only one single clause $\exists X.(a_1 \wedge \ldots \wedge a_n)$ containing only positive atoms, prefixed by some existential quantifiers.

S-Eq	$\exists X, x.(x \doteq y \wedge c)$	$\Rightarrow$	$\exists X.c\{x \mapsto y\}$	$(x \neq y)$
S-Feats	$\exists X, z.(x[f]y \wedge x[f]z \wedge c)$	$\Rightarrow$	$\exists X.(x[f]y \wedge c\{z \mapsto y\})$ $(y \neq z$, and if $z \in \mathcal{V}_o$ then $y \in \mathcal{V}_o)$	
S-Feats-Glob	$\exists X, x.(x[f]y \wedge x[f]z \wedge c)$	$\Rightarrow$	$\exists X, x.(x[f]y \wedge y \doteq z \wedge c)$	$(y, z \notin X)$
S-Sims	$x \sim_F y \wedge x \sim_G y \wedge c$	$\Rightarrow$	$x \sim_{F \cap G} y \wedge c$	
P-Feat	$x \sim_F y \wedge x[f]z \wedge c$	$\Rightarrow$	$x \sim_F y \wedge x[f]z \wedge y[f]z \wedge c$	$(f \notin F)$
P-Abs	$x \sim_F y \wedge x[f]\uparrow \wedge c$	$\Rightarrow$	$x \sim_F y \wedge x[f]\uparrow \wedge y[f]\uparrow \wedge c$	$(f \notin F)$
P-Fen	$x \sim_F y \wedge x[G] \wedge c$	$\Rightarrow$	$x \sim_F y \wedge x[G] \wedge y[F \cup G] \wedge c$	
P-Sim	$x \sim_F y \wedge x \sim_G z \wedge c$	$\Rightarrow$	$x \sim_F y \wedge x \sim_G z \wedge y \sim_{F \cup G} z \wedge c$ $(\text{if } \bigcap_{(y \sim_H z) \in c} H \not\subseteq F \cup G)$	

Fig. 3. Transformation rules for the positive case. Existential quantifiers are only written were relevant. Rule S-Feats is parameterized by a set $\mathcal{V}_o$ of variables.

In this subsection and the following, we will use transformation rules as the ones in Figure 3. These rules describe transformations that map a clause to a formula (in this subsection the resulting formula is also a clause, but that will no

longer be the case in the next subsection). We say that such a rule *left* $\Rightarrow$ *right* applies to a clause c if:

1. The pattern *left* matches the complete clause c modulo associativity and commutativity of conjunction.
2. The side conditions of the rule, if any, are met.
3. The transformation yields a formula which is *different* from c.

If c is a clause and r a transformation rule then we write $r(c)$ for the formula obtained by applying r to c.

Each of the rules of Figure 3 describes an equivalence transformation in the model $\mathcal{FT}$. Equation elimination (S-Eq) is a logical equivalence. S-Feats implements the fact that features are functional. This rule is parameterized by a set $\mathcal{V}_o$ of variables that will be the set of variables (local or global) of the input clause. The variable replacement is $\mathcal{V}_o$-oriented in the sense that we never replace a variable in $\mathcal{V}_o$ by a variable outside $\mathcal{V}_o$. S-Feat-Glob is similar to S-Feat for the case that y and z are both global variables. S-Sims allows us to contract multiple similarities between the same pair of variables into one. P-Feats, P-Abs and P-Fen propagate constraints along a similarity, taking into account the index of the similarity. Finally, P-Sim is a kind of transitivity of similarity, where we take care not to add a similarity which is subsumed by already existing similarities.

The propagations play two important roles in that system. First, they move information, possibly leading to a clash. This is the case in the following example where a fence moves through similarities to clash with a feature constraint:

$$\begin{array}{rl} & x[f]v \wedge x \sim_{\{g\}} y \wedge \underline{y \sim_{\{h\}} z} \wedge \underline{z[\varnothing]} \\ \text{P-Fen} & x[f]v \wedge \underline{x \sim_{\{g\}} y} \wedge \underline{y[\{h\}]} \wedge \underline{y \sim_{\{h\}} z} \wedge z[\varnothing] \\ \text{P-Fen} & x[f]v \wedge x[\{g,h\}] \wedge \underline{x \sim_{\{g\}} y} \wedge \underline{y[\{h\}]} \wedge y \sim_{\{h\}} z \wedge z[\varnothing] \end{array}$$

Second, they take information from local variables and move it to global variables. This mechanism is at the core of the elimination of existential quantifications, the idea being that once all the propagations took place, all interesting information is explicit in the global part, and we can hence drop the local part.

$$\begin{array}{rl} & y[h]\uparrow \wedge \exists z.(\underline{x \sim_{\{f\}} z} \wedge \underline{z \sim_{\{g\}} y}) \\ \text{P-Sim} & \underline{y[h]\uparrow} \wedge \underline{x \sim_{\{f,g\}} y} \wedge \exists z.(x \sim_{\{f\}} z \wedge z \sim_{\{g\}} y) \\ \text{P-Abs} & x[h]\uparrow \wedge y[h]\uparrow \wedge \underline{x \sim_{\{f,g\}} y} \wedge \exists z.(x \sim_{\{f\}} z \wedge z \sim_{\{g\}} y) \end{array}$$

The following function computes a normal form with respect to the rules of Figure 3:

```
function normalize-positive(c: positive clause)
  V_o := V(c_1) where c = ∃X.c_1
  while c does not clash and some rule r of Fig 3 applies to c:
    c := r(c)
  return(c)
```

Lemma 2. *For a positive clause c, the function* `normalize-positive` *terminates and yields a positive clause that is equivalent in $\mathcal{FT}$ to c.*

Given a quantifier-free clause c, we define D-part(c) as the conjunction of all D-literals of c.

Lemma 3. *Let the function* `normalize-positive` *return a clause $\exists X.c$ that does not clash and (g_c, l_c) be its decomposition. Let $d = D\text{-}elim(\exists X.D\text{-}part(c))$. If c contains no atom $x[f]y$ with $x \notin X$ and $y \in X$ then*

$$\mathcal{FT} \models \tilde{\forall}((\exists X.c) \leftrightarrow (g_c \wedge d))$$

Actually, both lemmas are special cases of the forthcoming Lemmas 5 and 6 of Section 4.3.

Lemma 3 can serve for quantifier elimination in the positive case, at least when there is no feature constraint from a global variable to a local one. We will see in Section 4.4 what can be done if this is not the case.

4.3 General Clauses with Local Variables

In case of clauses containing both positive and negative literals we have to consider transformation rules that introduce negations or disjunctions. However, our rules will continue to take a single clause as input. As a consequence, we have to transform the result obtained by a transformation into disjunctive normal form. We assume given a function `dnf` that takes a formula without universal quantifiers and containing only positive occurrences of existential quantifiers, and returns an equivalent dnf that does not contain any clashing clauses. This can be achieved by using a standard dnf transformation and then purging all clashing clauses, or alternatively by applying the clash rules on the fly.

Syntactic Sugar. In the transformation rules to be presented below we will use several abbreviations that allow us to write the rules more concisely. First we have

$$x\langle F\rangle \;:=\; \bigvee_{f\in F} \exists z.x[f]z$$

where $F \subset \mathcal{F}$ is a finite set. This formula states that x has *at least one* feature in the set F, it can be seen as a dual to the fence constraint $x[F]$ which states that x has *at most* the features in the set F. Note that $x\langle F\rangle$ introduces a disjunction, so introducing such a formula requires the result to be put into dnf.

The formula $x \not\doteq_f y$ states that x and y differ at feature f, that is either one of them has f and the other one does not, or their children at f are different. The formula $x \not\doteq_F y$ generalizes this to a finite set $F \subset \mathcal{F}$, stating that x and y differ at at least one of the features in F.

$$\begin{aligned} x \not\doteq_f y := {} & \exists z'.(x[f]\uparrow \wedge y[f]z') \vee \exists z.(x[f]z \wedge y[f]\uparrow) \\ & \vee \exists z, z'.(x[f]z \wedge y[f]z' \wedge (z \not\cong z' \vee z \not\sim_{\varnothing} z')) \\ x \not\doteq_F y := {} & \bigvee_{f\in F} x \not\doteq_f y \end{aligned}$$

We use $(z \not\cong z' \vee z \not\sim_\varnothing z')$ instead of $(z \not\doteq z)$ to denote a difference between two variables in order to avoid problems with the termination. These formulas introduce disjunctions. They also introduce negated similarities at some newly created children of x and y, so we have to take care in the termination proof when these formulas are introduced by a transformation.

R-NEQ	$x \not\doteq y \wedge c$	$\Rightarrow$	$(x \not\cong y \vee x \not\sim_\varnothing y) \wedge c$
R-NFEAT	$\neg x[f]y \wedge c$	$\Rightarrow$	$(x[f]\uparrow \vee \exists z.(x[f]z \wedge (y \not\cong z \vee y \not\sim_\varnothing z))) \wedge c$
R-NABS	$\neg x[f]\uparrow \wedge c$	$\Rightarrow$	$\exists z.x[f]z \wedge c$
R-NFEN-FEN	$x[F] \wedge \neg x[G] \wedge c$	$\Rightarrow$	$x[F] \wedge x\langle F \setminus G\rangle \wedge c$
R-NSIM-SIM	$x \sim_F y \wedge x \not\sim_G y \wedge c$	$\Rightarrow$	$x \sim_F y \wedge x \not\doteq_{F\setminus G} y \wedge c$
R-NSIM-FEN	$x[F] \wedge x \not\sim_G y \wedge c$	$\Rightarrow$	$x[F] \wedge \big(\neg y[F \cup G] \vee x \not\doteq_{F\setminus G} y\big) \wedge c$
E-NFEN	$x \sim_F y \wedge \neg x[G] \wedge c$	$\Rightarrow$	$x \sim_F y \wedge (\neg x[F \cup G] \vee x\langle F \setminus G\rangle) \wedge c$ $(F \not\subseteq G)$
E-NSIM	$x \sim_F y \wedge x \not\sim_G z \wedge c$	$\Rightarrow$	$x \sim_F y \wedge \big(x \not\sim_{F\cup G} z \vee x \not\doteq_{F\setminus G} z\big) \wedge c$ $(F \not\subseteq G)$

Fig. 4. Replacement and Enlargement rules for the general case. $\not\cong$ is the disequality of decorations.

New Rules. Figure 4 extends the previously defined set of rules by adding several replacement rules and two enlargement rules. First, we have R-NEQ, R-NFEAT and R-NABS that eliminate occurrences of the negated constraints $x \not\doteq y$, $\neg x[f]y$ and $\neg x[f]\uparrow$ respectively. Since no other rule introduces any of these negated constraints we can ignore these two negated constraints in the rest of the section.

Then we have three rules that combine a positive with a negative constraint. R-NFEN-FEN applies to the case where we have both a positive fence F and a negated fence G for x. We simplify this by keeping the positive fence F, and replacing the negative fence by saying that x must have a feature that is in F (since that is all it can have), but not in G. Similarly, R-NSIM-SIM applies when we have between x and y both a positive similarity except in F, and a negated similarity except in G. We simplify this by keeping the positive similarity, and replacing the negated similarity by stating that x and y differ at a feature that is in F (since these are the only features where they may differ) but not in G. Finally, R-NSIM-FEN applies when we have a fence F for x, and a negated similarity with y except in G. Note that for any F and G, $G^c = (F \cup G)^c \cup (F \setminus G)$. Hence, the negated similarity is equivalent to saying that either y has a feature outside $F \cup G$, which is the only possibility to have a difference with x outside $F \cup G$ since x has already fence F, or the difference is in the finite set $F \setminus G$.

Finally, we have the two enlargement rules E-NFEN and E-NSIM. Their sole purpose is to ensure (by enlarging the negated fence or the index of a negated similarity) that the rules in Figure 5 can be applied when we have a similarity in conjunction with a negated fence or a negated similarity. The correctness proof of

these rules is similar to the three previous rules. In fact, the similarity between x and y is not needed for the correctness of these two rules and serves only for the termination proof since the requirement of a context $x \sim_F y$ excludes arbitrary enlargements.

$$\begin{array}{lllll}
\text{P-NFEN} & x \sim_F y \wedge \neg x[G] \wedge c & \Rightarrow & x \sim_F y \wedge \neg x[G] \wedge \neg y[G] \wedge c & (F \subseteq G) \\
\text{P-NSIM} & x \sim_F y \wedge x \not\sim_G z \wedge c & \Rightarrow & x \sim_F y \wedge x \not\sim_G z \wedge y \not\sim_G z \wedge c & (F \subseteq G)
\end{array}$$

Fig. 5. Propagation rules for the general case.

The two rules in Figure 5 may propagate a negated fence or a negated similarity through a similarity. In fact, if x and y coincide outside F and $F \subseteq G$, then x and y also coincide outside G. Hence, if x has a feature outside G then so does y (P-NFEN), and if x differs from z at some feature outside G then so does y (P-NSIM).

We define the set of rules R_1 as the union of all the transformation rules of Figures 3 and 4, and R_2 as the set of the two transformation rules of Figure 5.

```
function normalize(c: clause)
  d := {c}
  V_o := V(c_1) where c = ∃X.c_1
  while exists c ∈ d to which some rule r ∈ R_1 applies
    d := (d \ {c}) ∪ dnf(r(c))
  while exists c ∈ d to which r ∈ R_2 applies
    d := (d \ {c}) ∪ {r(c)}
  return(d)
```

The function `normalize` normalizes first by rule set R_1, and then by rule set R_2. This decomposition is necessary to ensure termination. It also makes sense since application of rules R_2 conserves normal forms with respect to R_1.

Lemma 4. *The output of* ***normalize*** *is a dnf where each conjunction is in normal form for* $R_1 \cup R_2$.

Proof (sketch). We have to prove that the application of one of the rules in R_2 to a normal form with respect to R_1 does not produce a redex for any of the rules in R_1. Assume, for instance that the application of P-NFEN to c introduces a redex of R-NFEN-FEN. This means that the negative fence constraint introduced for y will react with a positive fence constraint (for y) that was already present in c. Since c is in normal form with respect to P-FEN, x must have a fence constraint in c. This yields a contradiction since then c is not in normal form with respect to R-NFEN-FEN. The other cases are similar (details can be found in appendix B).

Lemma 5. *The function* ***normalize****, when applied to a clause* c*, terminates and yields a dnf* d *such that* $\mathcal{FT} \models \tilde{\forall}(c \leftrightarrow d)$.

Proof (sketch). Equivalence of c and d follows from the fact that each transformation rule is an equivalence in $\mathcal{FT}$. Termination is shown by defining a well-founded order on clauses such that each rule transforms a clause into a set of stricter smaller clauses. The termination order on dnf formulas is the multiset extension [8] of this order.

This order is a lexicographic order over twelve different measures that decrease with the applications of the rules. We can for instance handle the rules R-NEq, R-NFeat and R-NAbs first by saying that they decrease the number of negated equalities, feature constraints or absences. Since nothing introduces those literals, this is already a good start.

The first main difficulty in finding that order comes from the fact that all the propagation rules are trying to saturate the clause. A good measure that decreases with them is then the set of all possible atoms that are not in the formula. For P-Feat, for instance: $\{(x[f]y) \mid x, y \in \mathcal{V}(c); f \in \mathcal{F}(c); (x[f]y) \notin c\}$. That would make a good measure if $\mathcal{V}(c)$ could not increase with the application of other rules such as R-NSim-Fen. We have thus to handle these other rules first, which leads us to another main difficulty.

The second main difficulty comes from the negated similarities. Indeed, while all other literals may only move "horizontally" following the similarities, negated similarities may "descend" in the constraint, creating variables and feature constraints if needed. It is not obvious when it will stop, and in particular to find a bound on the number of variables introduced.

Let us consider the following example constraint and one of its reduction paths (that is, the reduction may create several branches in the dnf, and we take only the one we are interested in):

$$x_0[f]x_1 \wedge x_1[f]y_0 \qquad \wedge\underline{x_0[\{f\}]} \wedge x_1[\{f\}] \wedge\underline{x_0 \not\sim_\varnothing y_0}$$

By R-NSim-Fen:

$$\exists y_1, \underline{z_1}.\ \underline{x_0[f]x_1} \wedge x_1[f]y_0 \wedge \underline{x_0[f]z_1} \wedge y_0[f]y_1 \wedge x_0[\{f\}] \wedge x_1[\{f\}] \wedge z_1 \not\sim_\varnothing y_1$$

By S-Feats:

$$\exists y_1.\quad x_0[f]x_1 \wedge x_1[f]y_0 \wedge y_0[f]y_1 \qquad \wedge x_0[\{f\}] \wedge x_1[\{f\}] \wedge x_1 \not\sim_\varnothing y_1$$

In two rules, we created a new variable y_1, and removed a negated similarity just to put it again somewhere else. Note in particular that R-NSim-Fen can still apply, because x_1 has now a fence and a negative similarity. In fact, if, instead of two, we take a number n of variables x_i, we can extend that example into one that always doubles the number of variables.

The key to our solution to this problem is that rules that make negative similarities descend, thus introducing feature constraints and new variables, need some "fuel", which is the presence of positive fences or similarities. We define the *original variables* as the variables that were in the clause at the begining of `normalize`. Then, we show that

1. the number of original variables cannot grow;
2. there are never feature constraints from non-original variables towards original ones;
3. the positive fences and similarities can only be present on original variables.

It remains the problem that negative similarities can descend. At some point, they will necessarily go too deep and leave the area where the original variables may live. By doing so, they loose the positive fences and similarities that they need to keep descending, and the process stops.

The full proof, including the lemmas corresponding to the points (1), (2) and (3), the definition of the measures and the technical details can be found in appendix A.

This is also where we make use of the quantifier elimination procedure for D-formulas. Given a quantifier-free clause c, we define D-part(c) as the conjunction of all D-literals of c.

Lemma 6. *Let the function* `normalize` *return a dnf which contains a clause* $\exists X.c$*. Let* (g_c, l_c) *be the decomposition of* c*, and* $d = D\text{-}elim(\exists X.D\text{-}part(c))$*. If* c *contains no atom* $x[f]y$ *with* $x \notin X$ *and* $y \in X$ *then*

$$\mathcal{FT} \models \tilde{\forall}((\exists X.c) \leftrightarrow g_c \wedge d)$$

The proof can be found in appendix B.

We call a clause *normalized* when it is an element of a dnf returned as result of function `normalize`.

4.4 Quantifier Elimination

In order to eliminate a block of existential quantifiers from a clause we apply iteratively the following rule:

$$\textsc{Feat-Fun} \quad \exists X, x.(y[f]x \wedge c) \quad \Rightarrow \quad \neg y[f]\uparrow \wedge \forall x.\,(y[f]x \rightarrow \exists X.\,(y[f]x \wedge c)) \qquad (y \notin X, y \neq x)$$

This rule follows the idea of [13], and was already applied to feature constraints in [17]. The correctness of this transformation is shown by the following chain of equivalences in the model $\mathcal{FT}$:

$\exists X, x.(y[f]x \wedge c)$	
$\exists x.(y[f]x \wedge \exists X.c)$	since $x, y \notin X$
$\neg y[f]\uparrow \wedge \forall x.\,(y[f]x \rightarrow \exists X.c)$	since features are functional
$\neg y[f]\uparrow \wedge \forall x.\,(y[f]x \rightarrow \exists X.\,(y[f]x \wedge c))$	

The last step is very important, because it ensures that, if $y[f]x \wedge c$ is in normal form, then the right part of the implication is also in normal form. This will be important for the function defined below.

The function `switch` defined below iterates this replacement for all local variables x that occur in the form $y[f]x$ where y is not local: the function applies the transformation Feat-Fun, and then recursively applies itself on the result. When there remains no more feature constraint $y[f]x$ from a global variable to a local variable in the normalized clause c, we meet the hypotheses of Lemma 6. We then return the conjunction of the global part g_c of the normalized clause, and of the D-part of the local part l_c from which we have eliminated the block of existential quantifiers.

```
recursive function switch(c: normalized clause)
  if ∃X, x.(y[f]x ∧ c′) matches c and y ∉ X:
    return(¬y[f]↑ ∧ ∀x.(y[f]x → switch(∃X.y[f]x ∧ c′)))
  else:
    (g_c, l_c) := decomposition(c)
    d := D-elim(∃X.D-part(l_c))
    return(g_c ∧ d)
```

Example 2. When given the following formula

$$\exists v, w.(y[f]v \wedge v[f]w \wedge w[f]z \wedge w[\{f, g\}] \wedge y \sim_{\varnothing} z)$$

the function `switch` returns

$$\neg y[f]\uparrow \wedge \forall v.(y[f]v \rightarrow (\neg v[f]\uparrow \wedge \forall w.(v[f]w \rightarrow (w[f]z \wedge y \sim_{\varnothing} z))))$$

Lemma 7. *Given a normalized clause c, `switch(c)` terminates and yields a formula ψ such that*

1. $\mathcal{FT} \models \tilde{\forall}(c \leftrightarrow \psi)$;
2. $\mathcal{V}(\psi) \subseteq \mathcal{V}(c)$;
3. *ψ contains no existential quantifiers and only positive occurrences of universal quantifiers;*
4. *If $\mathcal{V}(c) = \emptyset$ then ψ is quantifier-free.*

We can now write a function that transforms a Σ_1 formula into an equivalent Π_1 formula. For this we assume given a function `pnf` that transforms any formula into its prenex normal form.

```
function solve(p: Σ_1 formula)
  let ∃X.q = p where q is quantifier-free
  d := dnf(q)
  dt := ⋁_{c∈d} normalize(∃X.c)
  u := ⋁_{c∈dt} switch(c)
  return(pnf(u))
```

Finally, the function `decide` takes a formula in prenex normal form and returns an equivalent (in $\mathcal{FT}$) formula without any quantifiers. If Q is a string of quantifiers, then $\overline{Q}$ is the string of quantifiers obtained from Q by changing $\exists$ into $\forall$ and vice-versa. For instance, $\overline{\exists x \forall y \exists z} = \forall x \exists y \forall z$.

```
recursive function decide(p: pnf)
  if p is quantifier-free:
    return(p)
  else if p is Q.∃X.q
    where q quantifier-free, Q does not end on ∃:
    return(decide(Q. solve(∃X.q)))
  else if p is Q.∀X.q
    where q quantifier-free, Q does not end on ∀:
    return(¬ decide(Q̄. solve(∃X.¬q)))
```

Theorem 1. *Given a formula p in prenex normal form,* `decide(p)` *terminates and yields a formula q such that*

- $\mathcal{FT} \models \tilde{\forall}(p \leftrightarrow q)$
- $\mathcal{V}(q) \subseteq \mathcal{V}(p)$
- *q is a Π_1 formula, and quantifier-free in case $\mathcal{V}(p) = \emptyset$*

Proof (sketch). Termination follows from the fact that at each call to `decide`, the number of quantifier *alternations* in the pnf decreases.

If we apply `decide` to a closed formula, we hence obtain an equivalent (in $\mathcal{FT}$) formula that contains no free variables and no quantifiers. Since the only tree-terms are variables, we have obtained formula of the language D, for which we can decide by assumption validity in $\mathcal{D}$.

Corollary 1. *The first order theory of $\mathcal{FT}$ is decidable.*

5 Conclusion

We have presented a quantifier elimination procedure for a first-order theory of feature trees with similarity constraints. Since update constraints can be expressed by similarity and feature constraints, this implies in particular that the first-order theory of feature trees with update constraints is decidable.

Our model of feature trees is in several respects an abstraction of UNIX file systems [2]. First, real file systems make a distinction between different kinds of files (directories, regular files, various kinds of device files). This distinction is omitted here just for the sake of presentation. More importantly, real file systems are not really trees as they allow for multiple paths from the root to regular files (which must be sinks), and they provide for symbolic links. Since extending the model by any of these may lead to undecidability of the full first-order theory we might have to look for smaller fragments which are sufficient for our application to the symbolic execution of scripts.

Acknowledgments. The idea of investigating update constraints on feature trees originates from discussions with Gert Smolka a long time ago. We would like to thank the anonymous reviewers for their useful remarks and suggestions, and the members of the CoLiS project for numerous discussions on tree constraints and their use in modeling tree operations, in particular Claude Marché, Kim Nguyen, Joachim Niehren, Yann Régis-Gianas, Sylvain Salvati, and Mihaela Sighireanu.

References

1. Aït-Kaci, H., Podelski, A., Smolka, G.: A feature-based constraint system for logic programming with entailment. Theor. Comput. Sci. **122**(1–2), 263–283 (Jan 1994)
2. Bach, M.: The Design of the UNIX Operating System. Prentice-Hall (1986)
3. Backofen, R.: A complete axiomatization of a theory with feature and arity constraints. Journal of Logic Programming **24**(1&2), 37–71 (Jul/Aug 1995)

4. Backofen, R., Smolka, G.: A complete and recursive feature theory. Theor. Comput. Sci. **146**(1–2), 243–268 (Jul 1995)
5. Backofen, R., Treinen, R.: How to win a game with features. Information and Computation **142**(1), 76–101 (Apr 1998)
6. Comon, H., Lescanne, P.: Equational problems and disunification. Journal of Symbolic Computation **7**, 371–425 (1989)
7. Debian Policy Mailing List: Debian Policy Manual, version 4.1.3. Debian (Dec 2017), `https://www.debian.org/doc/debian-policy/`
8. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Communication of the ACM **22**(8), 465–476 (1979)
9. Hodges, W.: Model Theory, Encyclopedia of Mathematics and its Applications, vol. 42. Cambridge University Press (1993)
10. Jeannerod, N., Marché, C., Treinen, R.: A formally verified interpreter for a shell-like programming language. In: Paskevich, A., Wies, T. (eds.) Verified Software. Theories, Tools, and Experiments. LNCS, vol. 10712, pp. 1–18. Springer, Heidelberg, Germany (Jul 2017)
11. Jeannerod, N., Treinen, R.: Deciding the first-order theory of an algebra of feature trees with updates. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR'18. LNCS, Springer, Oxford, UK (Jul 2018)
12. Maher, M.J.: Complete axiomatizations of the algebras of finite, rational and infinite trees. In: LICS. pp. 348–357. IEEE, Edinburgh, Scotland, UK (Jul 1988)
13. Malc'ev, A.I.: Axiomatizable classes of locally free algebras of various type. In: Benjamin Franklin Wells, I. (ed.) The Metamathematics of Algebraic Systems: Collected Papers 1936–1967, chap. 23, pp. 262–281. North Holland (1971)
14. Smolka, G.: Feature constraint logics for unification grammars. Journal of Logic Programming **12**, 51–87 (1992)
15. Smolka, G., Treinen, R.: Records for logic programming. Journal of Logic Programming **18**(3), 229–258 (Apr 1994)
16. Treinen, R.: Feature constraints with first-class features. In: Borzyszkowski, A.M., Sokołowski, S. (eds.) Mathematical Foundations of Computer Science. LNCS, vol. 711, pp. 734–743. Springer (Aug/Sep 1993)
17. Treinen, R.: Feature trees over arbitrary structures. In: Blackburn, P., de Rijke, M. (eds.) Specifying Syntactic Structures, chap. 7, pp. 185–211. CSLI Publications and FoLLI (1997)
18. Vorobyov, S.: An improved lower bound for the elementary theories of trees. In: McRobbie, M.A., Slaney, J.K. (eds.) CADE'96. LNCS, vol. 1104, pp. 275–287. Springer, New Brunswick, NJ (Jul/Aug 1996)

A Proof of Lemma 5 (Termination)

A.1 Introduction and Example

To show the termination, we have to find a measure that decreases strictly at each step of the `while` loop of the function `normalize` of Section 4.3. When trying to find such an order one encounters quickly two problems:

– Some of the rules create new variables.

– Negated similarities cannot only be propagated horizontally (i.e., along similarities $x \sim_F y$) by rule P-NSIM, but also descend vertically (i.e., along feature constraints $x[f]y$), creating new feature constraints by R-NSIM-FEN, R-NSIM-SIM and E-NSIM.

Taken together, these two properties of the rule system pose a serious problem for termination, since we have to assure that pushing downwards of negated similarities terminates even when new variables and feature constraints may be created.

The idea of the proof is to show that the descent of negated similarities is in fact bounded. Let us first define a notion of depth of the variables in a clash-free constraint:

$$d_c(y) = \mathtt{max}\{1 + d_c(x) \mid (x[f]y) \in c\}$$

This definition of depth is valid because there are no cycles in the feature constraints in c. We want to show that, although the negated similarities are descending, they cannot go too far, and that the process stops.

One might hope to find a bound on the depths of the variables in a clause. However, it is not clear how this can be achieved. Here is an example where we create variables and almost double the depth of the clause:

$$\left(\bigwedge_{0 \le i < n} x_i[f]x_{i+1} \wedge x_i[\{f\}]\right) \wedge x_0 \not\sim_\varnothing x_n$$

Initially, all variables are at depth $\le n$. There is an occurrence of pattern $x_0[\{f\}] \wedge x_0 \not\sim_\varnothing x_n$. We can thus apply R-NSIM-FEN, and we get a set of clauses that contains the following:

$$\exists y_1, z_1. \left(\bigwedge_{0 \le i < n} x_i[f]x_{i+1} \wedge x_i[\{f\}]\right) \wedge x_0[f]z_1 \wedge x_n[f]y_1 \wedge z_1 \not\sim_\varnothing y_1$$

We have the pattern $\exists z_1. x_0[f]x_1 \wedge x_0[f]z_1$ and we can thus apply S-EQ which gives us:

$$\exists y_1. \left(\bigwedge_{0 \le i < n} x_i[f]x_{i+1} \wedge x_i[\{f\}]\right) \wedge x_n[f]y_1 \wedge x_1 \not\sim_\varnothing y_1$$

In two rule applications, we created a variable y_1 of depth $n+1$, and we reproduced the pattern consisting of a fence and a negated similarity that was on x_0 and x_n to x_1 and y_1. We can keep doing this, and we obtain in the end:

$$\exists (y_i)_{1 \le i \le n}. \left(\bigwedge_{0 \le i < n} x_i[f]x_{i+1} \wedge x_i[\{f\}]\right) \wedge x_n[f]y_1 \wedge \left(\bigwedge_{1 \le i < n} y_i[f]y_{i+1}\right) \wedge x_n \not\sim_\varnothing y_n$$

where the newly introduced variable y_n is at depth $2n$.

This shows that it is non trivial to have a bound on the depth of a clause, and on the number of variables in it. Luckily, there is a variant of this argument which we can prove. In fact, to descend and possibly create features, the negated similarities need "fuel", that fuel being the fences and positive similarities that are necessary to trigger R-NSim-Fen, R-NSim-Sim or E-NSim. We can show that this fuel can only be present on variables that were originaly present in the clause. And since their number cannot grow and their depth is bounded, we get what we want.

A.2 Technical Definitions

Let $\exists X_o.c_o$ be a clash-free clause. This will be the input clause to which we apply the function `normalize`. Define the original variables as:

$$\mathcal{V}_o = \mathcal{V}(c_o)$$

Note that $\mathcal{V}_o$ contains not only the free variables of the clause but also the variables of X_o that are present in c_o. In the following, we take S-Feats to be $\mathcal{V}_o$-oriented. Also, we will only consider clauses that are descendants of this clause c_o, that is, they are inhabitants at some point of the set d in the function `normalize`(c_o). In particular, they are all clash-free.

Let us define the depth of a variable in a clause c by:

Definition 1.

$$d_c(y) = \mathtt{max}\{1 + d_c(x) \mid (x[f]y) \in c\}$$

This definition is valid because c is clash-free, and in particular without cycles. We have two important properties about d_c:

Proposition 2. *For any clause $\exists X.c$, and any variable in $\mathcal{V}(c)$, we have that $d_c(x) < \mathtt{card}(\mathcal{V}(c))$.*

Proof. This is an immediate consequence of the fact that c is clash-free: no variable may appear twice on a path leading to x.

Proposition 3. *The depth of a variable cannot decrease. That is, for any clause $\exists X_1.c_1$ that transforms into $\exists X_2.c_2$ and any variable x in these two clauses, $d_{c_1}(x) \leq d_{c_2}(x)$.*

Proof. Consider clauses c_1 and c_2 and a variable x in both of them. There is a path in c_1 of length $d_{c_1}(x)$ leading to x by definition of the depth. We will show that this path is also present in c_2, although it may be slightly changed.

Our system contains several rules that add feature constraint leading to *new* variables, these rule do not modify any existing paths. There is only one rule that may change the existing feature constraints: S-Feats. However, it only renames one variable into an other, which means that all the features of our path are still present in the new clause. Also note that since $x \in \mathcal{V}(x_2)$, x cannot be the variable that is renamed. As a consequence, the path of length $d_{c_1}(x)$ in c_1 to x still leads to x in c_2, that is $d_{c_2}(x) \geq d_{c_1}(c)$.

We will now show several properties on the variables of the clauses $\exists X.c$ that descend from c_o.

Proposition 4. *For all $\exists X.c$ and $x \in \mathcal{V}(c) \setminus \mathcal{V}_o$, x is local in c, that is $x \in X$.*

Proof. The proof is induction on the rule sequence that was applied to obtain $\exists X.c$ from c_0. By inspecting the rules one verifies easily that any variable that is introduced by any rule also is existentially quantified.

We now have three important properties that state that positive equalities, similarities and fences cannot escape the set $\mathcal{V}_o$ of original variables. This means in particular that, although the negated similarities may descend, only the original variables have what is needed to trigger the rules R-NSim-Fen, R-NSim-Sim and E-NSim.

Proposition 5. *If $(x \doteq y) \in c$, then $x, y \in \mathcal{V}_o$.*

Proof. By induction on the rule sequence that led us there from c_o to c. This is trivially verified in c_o by definition of $\mathcal{V}_o$. The only rule that may introduce equalities is S-Feats-Glob, that only adds equalities between global hence original (Prop. 4) variables.

Proposition 6. *If $(x \sim_F y) \in c$, then $x, y \in \mathcal{V}_o$.*

Proof. By induction on the rule sequence that led us there from c_o to c. The only rules that may modify similarities are

1. S-Sims which only changes the index of an already existing similarity between the same variables,
2. P-Sim which creates a new similarity between two variables that each are already in a similarity relation,
3. S-Eq which may rename the variables in an existing similarity. However, it is only renaming original variables into original ones (Prop. 5).
4. S-Feats which may rename the variables in an existing similarity. However, if a renaming of say z into y introduced a similarity for variable $y \notin \mathcal{V}_o$ then we would have already a similarity for z, and $z \notin \mathcal{V}_o$ by the side condition of (S-Feats), which a is contradiction to the induction hypothesis.

Proposition 7. *If $(x[F]) \in c$, then $x \in \mathcal{V}_o$.*

Proof. By induction on the rule sequence that led us there from c_o to c. The only rules that may modify fences are

1. P-Fen which creates a fence constraint for a variable that appears in a similarity constraint and hence is, by Proposition 6, a variable of $\mathcal{V}_o$,
2. S-Eq which may rename the variable in an existing fence constraint. However, it only renames an original variable into an other original one.

3. S-Feats which may rename the variable in an existing fence constraint. However, if a renaming of say z into y introduced a fence constraint for variable $y \notin \mathcal{V}_o$ then we would have already a fence constraint for z, and $z \notin \mathcal{V}_o$ by the side condition of (S-Feats), which a is contradiction to the induction hypothesis.

We now want to prove that the depth of original variables is bounded by $\mathtt{card}(\mathcal{V}_o)$. However, new variables may have been introduced, and in order to prove our bound we have to assure that the introduction of new variables cannot increase the depth of original variables. The two following propositions will allow us to conclude that only original variables may occur on paths leading to original variables, which is sufficient to show that the depth of original variables cannot increase.

Let us first define the notion of fathers of a variable in a clause:

Definition 2.

$$\mathit{fathers}_c(y) = \{x \mid (x[f]y) \in c \text{ for some } f\}$$

Proposition 8. *Let $y \in \mathcal{V}(c)\backslash\mathcal{V}_o$. Then either $\mathit{fathers}_c(y) \subseteq \mathcal{V}_o$, or $\mathit{fathers}_c(y)$ is a singleton. In that second case, the only father is not in $\mathcal{V}_o$.*

Proof. This proposition is shown by induction on the rule applications from c_o that led to our constraint. The proposition is obviously true for c_o as there are no non-original variables in it. Let us now consider a step in the transformation, that is a rule that transformed a descendant $\exists X_1.c_1$ into $\exists X_2.c_2$. Assume that the proposition holds for c_1. There are several cases depending on the rule that constitutes our step.

- S-Eq transforms $c_1 = \exists x.(x \doteq y \wedge c)$ into $c_2 = c\{x \mapsto y\}$. From the side-condition of the rule and Prop. 5, we know that $x \neq y$ and that they are both original variables.
  The only variables whose fathers may have changed are y and the variables that had x for a father in c_1. y is not interesting for us here as it is an original variable.
  Say we have v non-original such that $x \in \mathtt{fathers}_{c_1}(v)$. Then $\mathtt{fathers}_{c_2}(v) = \mathtt{fathers}_{c_1}(v)\backslash\{x\}\cup\{y\}$. Since x is original, by induction hypothesis, $\mathtt{fathers}_{c_1}(v) \subseteq \mathcal{V}_o$. And since y is original, $\mathtt{fathers}_{c_2}(v) \subseteq \mathcal{V}_o$.
- S-Feats transforms $c_1 = \exists z.x[f]y \wedge x[f]z \wedge c$ into $c_2 = x[f]y \wedge c\{z \mapsto y\}$.
  The only variables whose fathers may have changed are y and the variables that had z for a father in c_1. Let us handle these two subcases:
  - If y is original, then there is nothing to prove. Let us assume it is not. Then, from the side-condition of S-Feats, we can tell that z is not original either. From the clash-freeness of c_1, we can deduce that y and z are not in $\mathtt{fathers}_{c_1}(y)$ nor $\mathtt{fathers}_{c_1}(z)$. Then $\mathtt{fathers}_{c_2}(y) = \mathtt{fathers}_{c_1}(y) \cup \mathtt{fathers}_{c_1}(z)$. Since y and z are both non-original, the induction hypothesis applies, and $\mathtt{fathers}_{c_1}(y)$ and $\mathtt{fathers}_{c_1}(z)$ are either both included in $\mathcal{V}_o$ or both singletons containing only x. In both cases, the invariant holds.

  - Let v be a non-original variable such that $z \in \texttt{fathers}_{c_1}(v)$. Then $\texttt{fathers}_{c_2}(v) = \texttt{fathers}_{c_1}(v) \setminus \{z\} \cup \{y\}$. From the induction hypothesis, we can say that either $\texttt{fathers}_{c_1}(v)$ is a singleton containing only z, in which case $\texttt{fathers}_{c_2}(v)$ is a singleton too and the invariant holds; or $\texttt{fathers}_{c_1}(v) \subseteq \mathcal{V}_o$. In that second case, we understand that z is original. From S-Feats's side-condition, we deduce that y has to be original too, and the invariant holds.
- P-Feat only adds one feature constraint $y[f]z$ in a constraint where there are $x[f]z$ and $x \sim_F y$, and thus only changes $\texttt{fathers}(z)$ with $\texttt{fathers}_{c_2}(z) = \texttt{fathers}_{c_1}(z) \cup \{y\}$. Since we have a similarity between them, x and y are originals (Proposition 6). But x is z's father in c_1, which means that $\texttt{fathers}_{c_1}(z) \subseteq \mathcal{V}_o$. Adding a new original father does not break the invariant: $\texttt{fathers}_{c_2}(z) \subseteq \mathcal{V}_o$.
- R-NFeat and R-NAbs both create one variable, but with only one father.
- R-NFen-Fen and E-NFen use the shortcut $x\langle F \rangle$ that contains a lot of variable introductions, but every time with only one father.
- R-NSim-Fen, R-NSim-Sim and E-NSim use the shortcut $x \not\simeq_F y$ that contains a lot of variable introductions, but every time with only one father.
- S-Sims, P-Abs, P-Fen, P-Sim, R-NEq, P-NFen and P-NSim do not introduce variables nor change the fathers of existing ones.

Now that we have proven this rather technical lemma, we can use it to prove the following, which is more interesting and leads directly to what we want:

Proposition 9. *If $y \in \mathcal{V}_o \cap \mathcal{V}(c)$, then $\texttt{fathers}_c(y) \subseteq \mathcal{V}_o$. In other words, there is no feature constraint from a non-original variable to an original variable.*

Proof. This proposition is shown by induction on the transformation from c_o that led to our clause. The proposition is obviously true for c_o as there are no non-original variables. Let us now consider a step in the transformation, that is a rule that transformed a descendant $\exists X_1.c_1$ into $\exists X_2.c_2$. Assume that the proposition holds for c_1. There are several cases depending on the rule that constitutes our step.

- S-Eq is non applicable because there are no equalities involving local variables in the constraints c_o and thus in c_1.
- S-Feats transforms $c_1 = \exists z.x[f]y \wedge xfz \wedge c$ into $c_2 = x[f]y \wedge c\{z \mapsto y\}$. From the side-condition and the clash-freeness of c_1, we know that $x \neq y$, $y \neq z$ and $x \neq z$. There are four sub-cases depending on whether y and z are originals.
  - if y and z are originals, we can freely replace z by y in the feature constraints without breaking the proposition for c_2.
  - if y and z are non-originals, there is by induction hypothesis no feature constraint from any of them to an original variable in c_1, so this is still the case in c_2.

- if y is original and z is non-original, then we obtain from the induction hypothesis that $\texttt{fathers}_{c_1}(y) \subseteq \mathcal{V}_o$. Since $x \in \texttt{fathers}_{c_1}(y)$, we have $x \in \mathcal{V}_o$. Since $x \in \texttt{fathers}_{c_1}(z)$ and $x \in \mathcal{V}_o$, we obtain by Proposition 8 that $\texttt{fathers}_{c_1}(z) \subseteq \mathcal{V}_o$. We conclude by $\texttt{fathers}_{c_2}(y) = \texttt{fathers}_{c_1}(y) \cup \texttt{fathers}_{c_1}(z)$.
- if y is non-original and z is original, then we are in contradiction with the side condition: this case cannot happen.

– P-Feat adds one feature constraint $y[f]z$ to a constraint where there are $x[f]z$ and $x \sim_F y$. y is original by Proposition 6. Adding a feature constraint $y[f]z$ where $y \in \mathcal{V}_o$ cannot invalidate the invariant.
– R-NFeat and R-NAbs both create one feature constraint but also introduce the necessary variable. We are thus sure that the proposition holds, as the introduced variable is non-original.
– R-NFen-Fen and E-Fen use the shortcut $x\langle F\rangle$ that contains a lot of feature constraints, but every time with a freshly created (thus non-original) variable.
– R-NSim-Fen, R-NSim-Sim and E-NSim use the shortcut $x \not\doteq_F y$ that contains a lot of feature constraints, but every time with a freshly created (thus non-original) variable.
– S-Sims, P-Abs, P-Fen, P-Sim, R-NEq, P-NFen and P-NSim do not introduce nor remove feature constraints.

As a consequence, no path leading to a original variable can contain a non-original variable. This means that the directed acyclic graph of the father relation has all the original variables in its top area, that is at a depth that is bounded by $\texttt{card}(\mathcal{V}_o)$. Combined with the fact that the fences and similarities do not leave this area, this will provide us with the weapons that we need to terminate our proof.

Proposition 10. *$d_c(x) < \texttt{card}(\mathcal{V}_o)$ if $x \in \mathcal{V}(c) \cap \mathcal{V}_o$.*

We now define the depth of atoms: it is the minimum of the depths of the variables involved. In particular, $d_c(x \not\sim_F y) = \texttt{min}(d_c(x), d_c(y))$.

Proposition 11. *$d_c(x \not\sim_F y) \leq \texttt{card}(\mathcal{V}_o)$*

Proof. We prove this property by induction on the transformation that led from c_0 to c. It is obviously true for c_0 because there are no non-original variables in c_0. Now, assume that the last step of the transformation leads from $\exists X_1.c_1$ to $\exists X_2.c_2$.

By induction hypothesis, we have for each negated similarity $x \not\sim_F y$ in c_1 only two possible cases: either $d_{c_1}(x \not\sim_F y) < \texttt{card}(\mathcal{V}_o)$ or $d_{c_1}(x \not\sim_F y) = \texttt{card}(\mathcal{V}_o)$.

1. If $(x \not\sim_F y) \in c_1$ with $d_{c_1}(x \not\sim_F y) = \texttt{card}(\mathcal{V}_o)$ then $d_{c_1}(x), d_{c_1}(y) \geq \texttt{card}(\mathcal{V}_o)$, hence $x, y \notin \mathcal{V}_o$ by Proposition 10. By Proposition 6 there is no similarity contraint for x or y in c_1, and by Proposition 7 there is no fence constraint for x or y in c_1. Hence, no rule producing a new negated similarity can apply to $x \not\sim_F y$.

2. If $(x \not\sim_F y) \in c_1$ with $d_{c_1}(x \not\sim_F y) < \texttt{card}(\mathcal{V}_o)$ then the negated similarity may either travel through a similarity with P-NSim, in which case it still touches an original variable and its depth stays smaller than $\texttt{card}(\mathcal{V}_o)$, or it can be rewritten with our rules R-NSim-Fen, R-NSim-Sim or E-NSim, in which case it reaches a higher depth. However, each of these rules can produced a negated similarity with depth at most $d_{c_1}(x \not\sim_F y)+1$. Since $d_{c_1}(x \not\sim_F y) < \texttt{card}(\mathcal{V}_o)$, each of the newly produced negated similarities has a depth in c_1 which is $\leq \texttt{card}(\mathcal{V}_o)$.

A.3 A Decreasing Measure

		S-Eq	S-Feats	S-Feats-Glob	S-Sim	P-Feat	P-Abs	P-Fen	P-Sim	R-NEq	R-NFeat	R-NAbs	R-NFen-Fen	R-NSim-Fen	R-NSim-Sim	E-NFen	E-NSim
1	Number of neg. eq.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
2	Number of neg. feat. constr.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
3	Number of neg. abs.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
4	Missing fences	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
5	Combined sims.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
6	Depth of neg. sims.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\not\geq$	$\not\geq$	$\cdot$	$\cdot$	$>$	$>$	$\cdot$	$\cdot$ $>$
7	Missing feats. in neg. sims.	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\not\geq$	$\not\geq$	$\cdot$	$\cdot$	$\not\geq$	$\not\geq$	$\cdot$	$>$ $\not\geq$
8	Missing feats. in neg. fences	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$>$	$\not\geq$	$\cdot$	$>$	$\cdot$
9	Missing equalities	$\geq$	$\geq$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$<$	$<$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$
10	Missing feature constraints	$\geq$	$\geq$	$<$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$<$	$<$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$
11	Missing absences	$\geq$	$\geq$	$\cdot$	$\cdot$	$\cdot$	$>$	$\cdot$	$\cdot$	$\cdot$	$<$	$<$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$	$\not\geq$
12	Number of literals	$>$	$>$	$\cdot$	$>$	$<$	$<$	$<$	$<$	$\cdot$	$\not\geq$	$\cdot$	$\cdot$	$\not\geq$	$<$	$\cdot$	$\not\geq$

1. Nothing introduces negative equalities, ever. The only rule acting on them is R-NEq that removes them.
2. Nothing introduces negative feature constraint. The only rule acting on them is R-NFeat that removes them.
3. Nothing introduces negative absences. The only rule acting on them is R-NAbs that removes them.
4. "Missing fences"

$$\{(x, F) \mid x \in \mathcal{V}(c) \cap \mathcal{V}_o;\ \text{no } x[F] \text{ in } c\}$$

This is the set of all the fences that could exist but that do not. This set clearly diminishes when we apply P-Fen. It is not touched by the rules that introduce variables, because those only introduce non original variables. It cannot increase when applying S-Eq or S-Feats, although those may remove may remove existing fences. However, when they do, this is because they removed the original variable to which the fence was attached.

5. "Combined similarities"

$$(x, y) \in (\mathcal{V}(c) \cap \mathcal{V}_o)^2 \mapsto \bigcap_{(x \sim_F y) \in c} F$$

We consider that a function is strictly smaller than an other if its domain is strictly included in the other's domain, or if their domain are equal, all values are smaller and at least one value is strictly smaller.
This function decreases with P-Sim but is left constant with S-Sim (it has been crafted precisely for that). Once again, if S-Eq or S-Feats were to remove a similarity and thus potentially change the values of these intersections, then an original variable would have disappeared, strictly decreasing the domain.

6. "Depth of the negated similarities" of c

$$\{\{\texttt{card}(\mathcal{V}_o) - d_c(x \not\sim_F y) \mid (x \not\sim_F y) \in c\}\}$$

We consider as an order for these multisets the lifting of the order in natural numbers: we allow ourselves to add a finite numbers to the multiset as long as we remove a number that is strictly greater than all of them. This is a well founded order if it is the lifting of a well founded order. And this is the case, because Prop. 11 tells us that all the numbers in it are non-negative.
This multiset is increased by R-NEq and R-NFeat because they add a new negative similarity (It would be increased by P-NSim too). It decreases in R-NSim-Fen and R-NSim-Sim because although we potentially add a negated similarity, we remove one the is higher (and whose depth is thus smaller). The case of E-NSim is a bit particular, because or measure stays constant in the first part of the disjunction while it decreases in the second part. The next measure will take care of that.

7. Missing features in negative similarities

$$\{\{\mathcal{F}(c) - F \mid (x \not\sim_F y) \in c\}\}$$

By definition, all the sets of the negated similarities are included in $\mathcal{F}(c)$.

8. Missing features in negative fences

$$\{\{\mathcal{F}(c) - F \mid (\neg x[F]) \in c\}\}$$

This multiset decreases with the rules R-NFen-Fen because this one removes an element, and with E-NFen because it either removes a negative fence or replace it by a larger one.

9. Missing equalities

$$\{(x, y) \mid (x \doteq y) \notin c\}$$

This set decreases with S-Feats-Glob. It increases a lot with all the rules adding variables, but they are no threat to us since we have already taken care of them.

10. Missing feature constraints

$$\{(x, f, y) \mid (x[f]y) \notin c\}$$

This set decreases with P-Feat.

11. Missing absences
$$\{(x, f) \mid (x[f] \uparrow) \notin c\}$$
This set decreases with P-Abs.
12. Number of literals
This number decreases with S-Eq and S-Feats.

B Proof of Lemma 6 (Elimination)

B.1 Normal Form

Lemma 8. *The output of* `normalize` *is a dnf where each conjunction is in normal form for R.*

We have to prove that P-NFen and P-NSim, when provided with a clause that is in normal form for the rules in R_1, conserves the normal form.

Let us take a constraint in normal form for R_1 and show that the application of one of the two rules in R_2 keeps it in normal form.

We will start with P-NFen. Clearly, since P-NFen only creates a new negative fence, the only rules it could trigger are R-NFen-Fen and E-NFen. Let us then take $c_1 = x \sim_F y \wedge \neg x[G] \wedge c$ that rewrites into $c_2 = x \sim_F y \wedge \neg x[G] \wedge \neg y[G] \wedge c$.

- R-NFen-Fen: c_1 is in normal form for R-NFen-Fen by hypothesis, so c cannot contain a positive fence for x. c_1 is also in normal form for P-Fen, so c cannot contain a positive fence for y either. Hence, c_2 is in normal form for R-NFen-Fen.
- E-NFen: c_1 is in normal form for P-Sim and S-Sim, so for each similarity $y \sim_H y'$ in c there is an I such that $x \sim_I y'$ is in c. For the same reason, we have that $H \subseteq F \cup I$. Since c_1 is in normal form for E-NFen, $F \subseteq G$ and $I \subseteq G$. Thus $H \subseteq G$, and c_2 is in normal form for E-NFen.

Let us continue with P-NSim. Since it only creates a negative similarity, the only rules it could trigger are R-NSim-Fen, R-NSim-Sim and E-NSim. Let us then take $c_1 = x \sim_F y \wedge x \not\sim_G z \wedge c$ that rewrites into $c_2 = x \sim_F y \wedge x \not\sim_G z \wedge y \not\sim_G z \wedge c$.

- R-NSim-Fen: c_1 is in normal form for R-NSim-Fen by hypothesis, so c cannot contain a positive fence for x. c_1 is also in normal form for P-Fen, so c cannot contain a positive fence for y either. Hence, c_2 is in normal form for R-NSim-Fen.
- R-NSim-Sim: c_1 is in normal form for R-NSim-Sim by hypothesis, so c cannot contain a positive similarity for (x, z). c_1 is also in normal form for P-Sim, so c cannot contain a positive fence for (y, z) either. Hence, c_2 is in normal form for R-NSim-Sim.
- E-NSim: c_1 is in normal form for P-Sim and S-Sim, so for each similarity $y \sim_H y'$ in c there is an I such that $x \sim_I y'$ is in c. For the same reason, we have that $H \subseteq F \cup I$. Since c_1 is in normal form for E-NSim, $F \subseteq G$ and $I \subseteq G$. Thus $H \subseteq G$, and c_2 is in normal form for E-NSim.

Thus, in the following, we can consider that the output of `normalize` is in normal form for all the rules.

B.2 Introduction to the Construction

Let $\exists X.c$ be a clause, that is an element of a dnf returned by the function `normalize`, and let (g_c, l_c) be the decomposition of c, that is in particular $\mathcal{V}(g_c) \cap X = \emptyset$. We also recall that D-part(l_c) is the part of l_c consisting of all its D-literals, and that $d = D$-elim$(\exists X.D$-part$(l_c))$ is a quantifier-free formula that is equivalent in $\mathcal{FT}$ to $\exists X.D$-part(l_c), due to Proposition 1.

We have to show that

$$\mathcal{FT} \models \tilde{\forall}(\exists X.(g_c \wedge l_c) \leftrightarrow (g_c \wedge d))$$

Since $\mathcal{V}(g_c) \cap X = \emptyset$ it is sufficient to show that

$$\mathcal{FT} \models \tilde{\forall}(\exists X.l_c \leftrightarrow d)$$

Since $\exists X.D$-part(l_c) is a logical consequence of $\exists X.l_c$, and since the formula $d = D$-elim$(\exists X.D$-part$(l_c))$ is equivalent in $\mathcal{FT}$ to the former, it only remains to show that

$$\mathcal{FT} \models \tilde{\forall}(d \rightarrow \exists X.l_c)$$

Let Y be the set of free variables of $\exists X.l_c$, which be assumption on the quantifier elimination procedure of $\mathcal{D}$ comprises the free variables of d, and assume a valuation $\rho_Y : Y \rightarrow \mathcal{FT}$. We will show that we can extend it to $\rho_{Y \cup X} : Y \cup X \rightarrow \mathcal{FT}$ such that $\mathcal{FT}, \rho_{Y \cup X} \models l_c$.

Note that l_c cannot contain a conjunction $x[f]y \wedge x[f]z$. Otherwise, we obtain a contradiction:

- If $y, z \notin X$ then (S-Feats-Glob) applies.
- If $y \in X$ and $z \notin X$, then $z \in \mathcal{V}_o$ (by Prop. 4 in App. A), hence (S-Feats) applies. The same reasoning applies when $y \notin X$ and $z \in X$.
- If $y, z \in X$ then (S-Feats) applies since we can choose the replacement such that the side condition is satisfied.

Recall that any clause in a dnf is clash-free. Since $\mathcal{FT}, \rho_Y \models d$, we have by Proposition 1 that $\mathcal{FT}, \rho_y \models \exists X.D$-part$(l_c)$. Hence, there is a variable assignment β such that $\mathcal{FT}, \rho_Y \cup \beta \models D$-part$(l_c)$.

In order to construct the extension of ρ_Y to X, choose a strict total order $\sqsubset$ over $Y \cup X$ such that

1. $y \sqsubset x$ whenever $(x[f]y) \in c$,
2. $y \sqsubset x$ whenever $y \in Y$ and $x \in X$.

This is possible because there are no feature cycles in c, due to clash pattern C-Cycle. Hence we can start with the partial order defined by $y \sqsubset x$ iff $x[f]y \in c$, and complete it into a total order by taking care to range all global variables before the local variables, which is possible due to the hypothesis of Lemma 6 that there is no feature from a global variable to a local variable.

B.3 Construction

We now define $\rho_{Y \cup X}$ by induction on X following $\sqsubset$ ensuring that we keep satisfied all the literals containing variables where our valuation is defined. The base case of the induction is ρ_Y, which is already defined and satisfies its literals by hypothesis. In the induction case, we take $x \in X$, we define the set of variables that are smaller than x, $Z = \{z \in Y \cup X \mid z \sqsubset x\}$ and we assume that we have ρ_Z already defined such that it satisfies all the literals about variables in Z.

We are going to extend it by defining $\rho_{Z \cup \{x\}}$. We define the following partial map m_x for x:

$$m_x = \{(f, \rho_Z(y)) \mid (x[f]y) \in c\}$$

Note that this defines a partial function, because there cannot be f, y and y' such that $(x[f]y) \in c$, $(x[f]y') \in c$ and $y \neq y'$ due to clash pattern S-Feats. Consider now the set of all the variables that are smaller than x and that are in a similarity relation with x:

$$down(x) = \{y \mid y \sqsubset x, (x \sim_H y) \in c \text{ for some } H\}$$

We will now define m'_x using m_x and $\rho_Z(y)$ for all the $y \in down(x)$. There are three cases depending on whether $down(x)$ is empty, and depending on whether there is a fence constraint for x.

1. If $down(x) = \emptyset$ and there is some fence constraint $(x[F]) \in c$, then we define $m'_x = m_x$.
2. If $down(x) = \emptyset$ and there is no fence constraint $(x[F]) \in c$, then we choose a fresh feature h_x which
   - does not occur in c (not even in a fence or similarity),
   - does not occur in $\mathtt{dom}(\overrightarrow{\rho_Y(y)})$, for any $y \in Y$,
   - is different from h_z, for any $z \sqsubset x$.

   This is possible since the mapping of a feature tree is required to have a finite domain, and since we have an infinite supply of feature symbols. Hence, the set
   $$\mathcal{F} \setminus \bigcup_{y \in Y} \mathtt{dom}(\overrightarrow{\rho_Y(y)})$$
   is infinite. Let $d \in \mathcal{D}$ be some arbitrary node decoration. We define $m'_x = m_x \cup \{(h_x, (d, \emptyset))\}$, that is m' is obtained from m by adding an edge labeled h_x going to the empty tree. This still defines a function because h_x is different from all the features encountered so far.
3. If $down(x) \neq \emptyset$ then we define
   $$m'_x = m_x \cup \bigcup_{\substack{z \sqsubset x \\ (x \sim_H z) \in c}} \overrightarrow{\rho_Z(z)} \restriction_{H^c}$$
   where, when $\overrightarrow{\rho_Z(z)} \restriction_{H^c}$ is the restriction of $\overrightarrow{\rho_Z(z)}$ to the complement of H in $\mathcal{F}$. This union is not disjoint, so we have to show that m'_x is well-defined as a function.

(a) Assume that $f \in \mathtt{dom}(m_x)$ and $f \in \mathtt{dom}(\overrightarrow{\rho_Z(z)} \restriction_{H^c})$, with $z \sqsubset x$ and $(x \sim_H z) \in c$. By definition of m_x, there must be a $(x[f]y) \in c$, with $y \sqsubset x$ and $m_x(f) = \rho_Z(y)$.
Since $f \in \mathtt{dom}(\overrightarrow{\rho_Z(z)} \restriction_{H^c})$, we have that $f \notin H$. By rule P-Feat, It must be the case that $(z[f]y) \in c$. Since $y, z \sqsubset x$, we obtain by induction hypothesis that $\mathcal{FT}, \rho_Z \models z[f]y$, that is

$$\overrightarrow{\rho_Z(z)} \restriction_{H^c} (f) = \overrightarrow{\rho_Z(z)}(f) = \rho_Z(y)$$

(b) Assume that $f \in \mathtt{dom}(\overrightarrow{\rho_Z(z)} \restriction_{H^c})$ and $f \in \mathtt{dom}(\overrightarrow{\rho_Z(z')} \restriction_{H'^c})$, with $z, z' \sqsubset x$, $(x \sim_H z) \in c$, $(x \sim_{H'} z') \in c$, and $z \neq z'$ or $H \neq H'$.
By rule P-Sim, there is some $I \subseteq H \cup H'$ such that $(z \sim_I z') \in c$. Since $f \notin H, H'$, we also have that $f \notin I$. Since $z, z' \sqsubset x$, we have by induction hypothesis that $\mathcal{FT}, \rho_Z \models z \sim_I z'$. Since $f \notin I$, this means that

$$\overrightarrow{\rho_Z(z)} \restriction_{H^c} (f) = \overrightarrow{\rho_Z(z)}(f) \restriction_{I^c} (f) = \overrightarrow{\rho_Z(z')}(f) \restriction_{I^c} (f) = \overrightarrow{\rho_Z(z')} \restriction_{H'^c} (f)$$

Finally, we define

$$\rho_{Z \cup \{x\}}(z) = \begin{cases} (\beta(x), m'_x) & \text{if } z = x \\ \rho(z) & \text{if } z \sqsubset x \end{cases}$$

B.4 Verification

Let us now show that that the invariant is satisfied, that is that $\mathcal{FT}, \rho_{Z \cup \{x\}} \models l$ for every $l \in c$ such that $z \sqsubset x$ for every $z \in \mathcal{V}(c)$. By induction hypothesis, we may restrict ourselves to the case where $x \in \mathcal{V}(c)$. We distinguish the different possible forms of a literal c:

$x \doteq y$ By rule S-Eq, this is only possible when $x = y$ or when x and y are global. The first case is always trivially satisfied, the second is satisfied by ρ_Y by hypothesis.

$x \not\doteq y$ Eliminated by the system (R-NEq).

$x[f]y$ This is immediate considering the way m'_x was defined.

$\neg x[f]y$ Eliminated by the system (R-NFeat).

$x[f] \uparrow$ We distinguish the three cases in the construction of m'_x:

1. There cannot be a literal $(x[f]y) \in c$, thanks to rule (C-Feat-Abs). Thus, m_x is not defined for f.
2. In addition to case (1), note that $f \neq h_x$ due to the way h_x was chosen. Thus, m'_x is not defined for f.
3. For all $z \sqsubset x$ with $(x \sim_H z) \in c$, if $f \notin H$, then $(z[f] \uparrow) \in c$ (P-Abs). Since this last atom is satisfied by induction hypothesis, $f \notin \mathtt{dom}(\overrightarrow{\rho_Z(z)})$. That, combined with the point (1) gives us: $f \notin \mathtt{dom}(\overrightarrow{\rho_{Z \cup \{x\}}(x)})$.

$\neg x[f] \uparrow$ Eliminated by the system (R-NAbs).

$x[F]$ We distinguish the three cases in the construction of $\rho(x)$:

1. There cannot be a literal $(x[f]y) \in c$ with $f \notin F$ for x (C-Feat-Fen). Thus, $\mathtt{dom}(m_x) \subseteq F$.
2. Does not apply.
3. In addition to case (1), for all $z \sqsubset x$ with $(x \sim_H z) \in c$, we have that $(z[H \cup F]) \in c$ (P-Fen. Since it is satisfied by ρ_Z by induction hypothesis, $\mathtt{dom}(\overrightarrow{\rho_Z(z)}) \subseteq H \cup F$, and hence $\mathtt{dom}(\overrightarrow{\rho_Z(z)} \restriction_{H^c}) \subseteq F$. This, together with the reasoning of case (1), give us $\mathtt{dom}(\overrightarrow{\rho_{Z\cup\{x\}}(x)}) \subseteq F$.

$\neg x[F]$ We distinguish the three cases in the construction of $\rho(x)$.

1. This rule does not apply: By rule R-NFen-Fen, we cannot have both a positive and negative fence constraint for the same variable.
2. In this case, $\mathtt{dom}(\overrightarrow{\rho_{Z\cup\{x\}}(x)})$ contains the fresh feature $h_x \notin F$.
3. In this case there is a variable $z \sqsubset x$, such that $(x \sim_H z) \in c$. By E-NFen we must have that $H \subseteq F$ (Note that this rule generates several alternatives: one containing an enlarged negative fence, and the other ones where the negative fence is replaced by an absence constraint. Hence, in presence of the negative fence, we must be in the first alternative). Then, by P-NFen, $(\neg z[F]) \in c$. By induction hypothesis, $\mathcal{FT}, \rho_Z \models \neg z[F]$, that is there is $f \in \mathtt{dom}(\overrightarrow{\rho_Z(z)})$ with $f \notin F$, hence $f \notin H$. By construction of m'_x this means that $f \in \mathtt{dom}(\overrightarrow{\rho_{Z\cup\{x\}}(x)})$, that is $\mathcal{FT}, \rho_{Z\cup\{x\}} \models \neg x[F]$.

$x \sim_F y$ Satisfied by construction.

$x \not\sim_F y$ We distinguish the three cases in the construction of $\rho(x)$.

1. This case does not apply, since by R-NSim-Fen, x cannot have both a negated similarity and a fence constraint.
2. By construction, there is a fresh feature $h_x \in \mathtt{dom}(\overrightarrow{\rho_{Z\cup\{x\}}(x)})$ Since $y \neq x$ (C-NSim-Refl) and since there is no similarity between x and y (R-NSim-Sim), we have that $h_x \notin \mathtt{dom}(\overrightarrow{\rho_Z(y)})$. Since $h_x \notin F$, this means that $\mathcal{FT}, \rho_{Z\cup\{x\}} \models x \not\sim_F y$.
3. In this case there is variable $z \sqsubset x$ and H such that $(x \sim_H z) \in c$. As we have seen in the previous case, this means that

$$\mathcal{FT}, \rho_{Z\cup\{x\}} \models x \sim_H z \tag{1}$$

By (E-NSim) we must have that $H \subseteq F$ (by the same reasoning as above, we must be in the first of the alternatives introduced by this rule). Then, by rule (P-NSim), $H \subseteq F$ and $(z \not\sim_F y) \in c$. By induction hypothesis,

$$\mathcal{FT}, \rho_Z \models z \not\sim_F y \tag{2}$$

Since $H \subseteq F$, it follows from (1) that

$$\mathcal{FT}, \rho_{Z\cup\{x\}} \models x \sim_F z \tag{3}$$

Finally, we conclude from (3) and (2) that

$$\mathcal{FT}, \rho_{Z\cup\{x\}} \models x \not\sim_F y \tag{4}$$

C Optimisation

We can add a few simplification rules that do not affect the outcome of the algorithm nor its properties. See Fig. 6.

S-Eq-Glob	$x \doteq y \wedge c$	$\Rightarrow$	$x \doteq y \wedge c\{x \mapsto y\}$	$(x, y \in \mathcal{V}(c))$
S-Eq-Refl	$x \doteq x \wedge c$	$\Rightarrow$	c	
S-Sim-Refl	$x \sim_F x \wedge c$	$\Rightarrow$	c	
S-Fens	$x[F] \wedge x[G] \wedge c$	$\Rightarrow$	$x[F \cap G] \wedge c$	
S-Fen-Abs	$x[F] \wedge x[f] \uparrow \wedge c$	$\Rightarrow$	$x[F \setminus \{f\}] \wedge c$	
S-NFen-Abs	$\neg x[F] \wedge x[f] \uparrow \wedge c$	$\Rightarrow$	$\neg x[F \cup \{f\}] \wedge c$	
S-NFen-Feat	$\neg x[F] \wedge x[f]y \wedge c$	$\Rightarrow$	$x[f]y \wedge c$	$(f \notin F)$
S-NFens	$\neg x[F] \wedge \neg x[G] \wedge c$	$\Rightarrow$	$\neg x[G] \wedge c$	$(F \subseteq G)$
S-NSims	$x \not\sim_F y \wedge x \not\sim_G y \wedge c$	$\Rightarrow$	$x \not\sim_G y \wedge c$	$(F \subseteq G)$

Fig. 6. Optional simplification rules.

D Examples

Let us apply `decide` to the following formula:

$$\forall x \cdot \exists y, z \cdot (x[f] \uparrow \vee (x \sim_g y \wedge y[f]z \wedge \texttt{newer}(x, y))) \tag{D.1}$$

This formula is of the form $Q \cdot \exists X \cdot q$ where q is quantifier-free and Q does not end with $\exists$. We thus need to apply `solve` to

$$\exists y, z \cdot (x[f] \uparrow \vee (x \sim_g y \wedge y[f]z \wedge \texttt{newer}(x, y))) \tag{D.2}$$

and to replace it (D.2) by the result in D.1. In order to do that, we put D.2 in disjunctive normal form and obtain:

$$\begin{array}{l} \exists y, z \cdot x[f] \uparrow \\ \vee\ \exists y, z \cdot (x \sim_g y \wedge y[f]z \wedge \texttt{newer}(x, y)) \end{array} \tag{D.3}$$

The first clause is really simple and does not really change when we apply `normalize` or `switch`, let us focus on the second one and apply `normalize`:

$$\exists y, z \cdot (x \sim_g y \wedge x[f]z \wedge y[f]z \wedge \texttt{newer}(x, y)) \tag{D.4}$$

We now apply `switch`. That will first change the existential quantifier for z because of the feature $x[f]z$ and give:

$$\neg x[f] \uparrow \wedge \forall z \cdot (x[f]z \to \exists y \cdot (x \sim_g y \wedge x[f]z \wedge y[f]z \wedge \texttt{newer}(x, y))) \tag{D.5}$$

`switch` then applies the result of Lemma 6, considering that the elimination of quantifiers on decorations gives us $(\exists y \cdot \texttt{newer}(x, y)) \leftrightarrow \texttt{newer}_0(x)$ stating that x's timestamp is greater than 0. The result of `switch` is thus:

$$\neg x[f]\uparrow \wedge \forall z \cdot (x[f]z \rightarrow \texttt{newer}_0(x)) \tag{D.6}$$

or, if we replace the implication

$$\neg x[f]\uparrow \wedge \forall z \cdot (\neg x[f]z \vee \texttt{newer}_0(x)) \tag{D.7}$$

After putting everything in prenex normal form, the result of `solve` is thus:

$$\forall z \cdot \begin{array}{c} x[f]\uparrow \\ \vee\ \neg x[f]\uparrow \wedge (\neg x[f]z \vee \texttt{newer}_0(x)) \end{array} \tag{D.8}$$

This is one step of the `decide` function that removes a quantifier alternation when the last bloc is made of existential quantifiers. We now have to decide:

$$\forall x, z \cdot x[f]\uparrow \vee (\neg x[f]\uparrow \wedge (\neg x[f]z \vee \texttt{newer}_0(x))) \tag{D.9}$$

and, to do that, we need to decide:

$$\exists x, z \cdot \neg x[f]\uparrow \wedge (x[f]\uparrow \vee (x[f]z \wedge \neg\texttt{newer}_0(x))) \tag{D.10}$$

We apply `solve` that puts this formula in disjunctive normal form:

$$\exists x, z \cdot \begin{array}{c} \neg x[f]\uparrow \wedge x[f]\uparrow \\ \vee\ \neg x[f]\uparrow \wedge x[f]z \wedge \neg\texttt{newer}_0(x) \end{array} \tag{D.11}$$

It then normalizes, which deletes the first clause and gives:

$$\exists x, z \cdot (x[f]z \wedge \neg\texttt{newer}_0(x)) \tag{D.12}$$

The function `switch` then applies Lemma 6 that removes $x[f]z$. The quantifier elimination in decorations gives us $(\exists x \cdot \neg\texttt{newer}_0(x)) \leftrightarrow \top$, and the whole formula reduces to $\top$. Since there was a negation in `decide`, we conclude that D.1 is equivalent to $\bot$.