An Extension of Explanation-Based Generalization to Negation as Failure

Stefan Schrödl[†]

December 6, 1994

Abstract

Implementations of *Explanation-Based Generalization (EBG)* within a logic-programming environment, as e.g. the well-known PROLOG-EBG algorithm [KCMcC87], are able to generalize the proof of a goal from a *definite* (i.e. Horn clause) domain theory. However, it is a fact that practical applications frequently require the enhanced expressiveness of *negations* in rule bodies. Specifically, this is the case for the domain of *game playing*, where traditional EBG has turned out to be inadequate [Ta89]. In this paper we present an approach which extends EBG to this more general setting; it is described in the form of a transformation system, and comprises Siqueira and Puget's method of *Explanation-Based Generalization of Failures* [SiPu88] for definite programs. For the case that both domain theory and training example are represented as *allowed* normal programs, we prove that the derived clause satisfies the standard requirements of EBG, namely *operationality*, *sufficiency*, and *correctness*. Furthermore, a meta-interpreter implementation is given.

Keywords: Explanation-Based Generalization, Negation as Failure, Logic Programming, Program Transformation, Meta-Interpreter, Game Playing

[†]Institut für Informatik, Am Flughafen 17, D-79110 Freiburg, Germany; e-mail: schroedl@informatik.uni-freiburg.de

1 Introduction

Explanation-Based Generalization (EBG) is an analytical learning paradigm usually defined in the following way [MKKC86].

Given:

a *target concept*, describing the concept being learned,

a *training instance* of the target concept,

a *domain theory*, a set of facts and rules about the domain,

an *operationality criterion*, specifying the form of the learned concept definition,

Determine:

A generalization of the training example that is a sufficient definition of the target concept and that satisfies the operationality criterion.

EBG has been widely studied in a logic programming environment (cf. [Hi87], [KCMcC87]): Typically, both the domain theory P_{nop} and the training example P_{op} are represented in a uniform definite input program P_{in}. The explanation is identified with a proof for a goal of a specified predicate symbol, the target concept, where the proof method is the standard procedural concept of *SLD-resolution* [Ll87]. The operationality criterion reduces to a list of the predicate names which are allowed to appear in the derived rule body; these are generally the same as those occurring in the training example P_{op}.

The underlying assumption is that the operational part P_{op} of the input program P_{in} is frequently exchanged with different examples P'_{op} and used together with the non-operational part P_{nop} remaining fixed. Therefore, P_{op} and P'_{op} do not depend on predicates defined in P_{nop}, such that they can safely be separated. By adding the learned (redundant) rule to the domain theory, different examples of the same proof structure can be recognized without (possibly expensive) recourse to the domain theory.

Several authors have pointed out that it is also useful to learn from counterexamples (cf. [Gup87], [MiCa87], [MoBha87]): From the failed proof tree, a condition is to be found which is satisfied by the instance, and is sufficient to ensure the *failure* of the goal. Such a condition may prevent needless repetitions of failing proof attempts. A systematic solution was proposed by Siqueira and Puget in [SiPu88], which they called *Explanation-Based Generalization of Failures (EBGF)*.

Both the techniques of EBG and EBGF are restricted to *definite* domain theories; this implies that clause bodies may only contain positive literals. However, each PROLOG programmer is familiar with the fact that definite programs, although Turing-equivalent, frequently lack sufficient expressiveness for many practical situations. The standard procedural concept of *SLDNF-resolution*, which is based on the *Negation-as-(Finite)-Failure*-rule [Cla78], can be used to allow negative literals in goals and in bodies of program clauses, which will be called *normal* in the following [Ll87].

Considerable research has been devoted to Explanation-Based Learning in the domain of *game playing*; efficient evaluation functions can be discovered by re-expressing implicitly defined concepts of win and loss in a language of "immediately observable" features, e.g. in terms of spatial relations between figures on the board. However, existing systems are generally not able to deduce sufficient and correct classification rules for arbitrary game positions: Part of them, as e.g. Minton's *Constraint-Based Generalization* [Min84], deduce correct recognition rules for different games, but are limited to certain classes of positions (in this case to those for which a forcing strategy exists). Others (e.g. Flann and Dietterich [FlaDi86], Tadepalli [Ta89], and Yee et al. [YSUB90]) derive incorrect conditions which are subsequently refined by different methods. The *intractable theory problem* [Ta89] states that traditional EBG is not directly applicable, although a complete axiomatization can be provided. We conjecture that one of the underlying reasons is the inherent need for some kind of negation. Indeed, the general concept of a game *Position* which is lost for the *Player* to move in at most *N* plies can be expressed most naturally in the following way:

```
lost(Position,Player,_) ←
    game_over(Position,Player).
lost(Position,Player,N) ←
    N > 0, N1 is N − 1,
    opponent(Player,Opponent),
    not (move(Player,Position,PositionAfter),
        not won(PositionAfter,Opponent,N1)).
```

```
won(Position,Player,N) ←
      N > 0, N1 is N − 1,
      opponent(Player,Opponent),
      move(Player,Position,PositionAfter),
      lost(PositionAfter,Opponent,N1).
```

Consequently, Siqueira and Puget's work suggests itself for an evident extension: Explanation-Based Generalization of successful proofs for goals from *normal* programs. The present report addresses this question, which, to the best of our knowledge, has not been treated previously. It is organized as follows:

Section 2 formalizes EBG as a clause transformation task, in analogy to *Partial Evaluation*. This view facilitates the presentation of our method, since it allows to subdivide it into five basic transformation rules. These are introduced in section 3: Section 3.1 recalls the basic operation of traditional EBG for definite programs, as seen in this framework. The transformation of section 3.2 essentially corresponds to EBGF. Section 3.3 defines a simplification step which eliminates unifications. For normal programs, two more operations are required, which are treated in sections 3.4 and 3.5: they cover the cases of nested negations, as well as of negative subgoals in which operational literals are selected. Section 4 starts with a formalization the standard requirements of EBG, namely *operationality*, *sufficiency*, and *correctness* of the derived clause (4.1); its remainder consists of a proof that the presented transformation system satisfies this specification, under the assumption that both domain theory and training example are *allowed* normal programs. An implementation of the method in the form of a meta-interpreter is given in section 5.

2 EBG as a clause transformation task

As was pointed out in [HaBu88] and further explored in [BruSch89], there is a close relation between EBG and the program transformation technique of *Partial Evaluation (PE)*, since both are based on the *unfolding* operation [TaSa84]. Usually PE is presented as a sequence of such basic steps.

It will be convenient to also decompose EBG into the sequence of elementary computation steps performed by a procedure such as PROLOG-EBG

[KCMcC87]. Each such operation transforms an intermediate clause representing the point of computation reached so far. This mode of description has the advantage of *modularity*, since each transformation rule can be treated independently. Furthermore, it is possible to conceive and integrate additional transformations different from those considered here.

This idea can be formalized in the following way: Let the n-ary predicate symbol p defined in P_{nop} be the target concept, and G be an atom of this predicate; suppose an example SLDNF-refutation R_1 for $P_{in} \cup \{\leftarrow G\}$ is given, with a computed answer σ_1. For the moment, it suffices to think of P_{in} as being simply the union of P_{nop} and P_{op}; later on, we will add clauses for two auxiliary predicates, and also use a slightly more general notion of a refutation.

We always start with a trivial clause $p(x_1, \ldots, x_n) \leftarrow p(x_1, \ldots, x_n)$. A *transformation sequence* for $(G\sigma_1, P_{nop}, P_{op})$ consists of a sequence of clauses $C_1, \ldots, C_n$ and a sequence of refutations $R_1, \ldots, R_n$, where each C_i is derived from C_{i-1} by an elementary transformation step guided by the example refutation R_{i-1}. Therefore, for each C_i which has the form $H_i \leftarrow B_i$, R_i is a refutation for $P_{in} \cup \{\leftarrow B_i\}$, and H_i, instantiated with the answer substitution σ_i computed by R_i, must be at least as general as $G\sigma_1$.

Definition 1 (Transformation Sequence) *Suppose P_{nop} and P_{op} are normal programs, and A is an atom of the form $p(t_1, \ldots, t_n)$. Let P_{in} be the normal program $P_{nop} \cup P_{op}$, and A' be the atom $p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are distinct new variables not occurring in $P_{in} \cup \{\leftarrow A\}$.*

A transformation sequence *for (A, P_{nop}, P_{op}) consists of a sequence of normal clauses $C_1, \ldots, C_n$ and a sequence of SLDNF-refutations $R_1, \ldots, R_n$, where C_1 is $A' \leftarrow A'$, and for each $i \in \{1, \ldots, n\}$,*

- *R_i is a refutation for $P_{in} \cup \{\leftarrow B_i\}$, where C_i has the form $H_i \leftarrow B_i$; moreover, if σ_i is the answer substitution for B_i computed by R_i, then $H_i\sigma_i\mu = A$, for some μ.*
- *For $i > 1$, C_i is the result of a transformation $T(C_{i-1}, P_{nop}, P_{op}, R_{i-1})$.*

We call C_n the result *of the transformation sequence.*

At first glance, it might be confusing that we refer to *different* refutations in the transformation sequence. However, R_i can always be constructed by modifying R_{i-1} in such a way that it preserves the information which is relevant for the generalization of that part of the clause which is left unchanged by the transformation step. For traditional EBG, this information consists of the program clauses used to resolve the atoms in the body; as we will see in section 4, R_{i-1} is turned into R_i according to the switching lemma [Ll87]. Hence, the transformation sequence can derive the same resulting clause as the usual procedural definition of EBG.

3 Transformation steps for normal programs

3.1 Traditional EBG: Reducing non-operational atoms

In this section we will introduce five transformation rules in turn which constitute our system. The first one, called T_{nop}, corresponds to the basic computation step in traditional EBG: it eliminates non-operational atoms. Besides a clause C to be transformed, it takes as arguments a domain theory P_{nop}, a training example P_{op} and a refutation R to be generalized. The use of R is the distinguishing feature from the unfolding operation in PE: It guides the choice of the program clause used for unfolding C at a non-operational literal L_k, where the concrete instantiation in the example is abstracted away [BruSch89].

Transformation 1 (T_{nop}) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op}$, C be a normal clause of the form $H \leftarrow L_1, \ldots, L_m$. Suppose that R is an SLDNF-refutation for $P_{in} \cup \{\leftarrow L_1, \ldots, L_m\}$ with a sequence of goals $G_0, G_1, \ldots, G_r = \Box$, input clauses $Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$.*

A clause C^T is the result of transformation $T_{nop}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *For some $k \in \{1, \ldots, m\}$, L_k is an atom whose predicate symbol is defined in P_{nop}.*
- *G_l ($l \in \{0, \ldots, r-1\}$) is the goal in which (an instance of) L_k is selected, and Q_{l+1} has the form $A \leftarrow A_1, \ldots, A_n$.*
- *θ is an mgu for L_k and A, and C^T is*
$$(H \leftarrow L_1, \ldots, L_{k-1}, A_1, \ldots, A_n, L_{k+1}, \ldots, L_m)\theta.$$

3.2 Reducing non-operational literals within negated subgoals

Throughout this section, the transformation process will be illustrated using the following example:

P_{nop} :	$p(x) \leftarrow q(x), \sim r(x).$	P_{op} :	$q(a) \leftarrow .$
	$r(x) \leftarrow v(x), \sim w(x).$		$v(a) \leftarrow .$
	$r(x) \leftarrow s(x,y), \sim t(y), \sim u(y).$		$w(a) \leftarrow .$
	$t(b) \leftarrow .$		$s(a,b) \leftarrow .$
	$u(c) \leftarrow .$		$s(a,c) \leftarrow .$

The target concept is $p/1$, $P_{in} \cup \{\leftarrow p(x_1)\}$ has an SLDNF-refutation R_1 with computed answer $\{x_1/a\}$. For ease of exposition, if we do not mention a refutation R_i, then it is assumed that the PROLOG computation rule is used, which always selects the leftmost literal in a goal. We start with a trivial clause $C_1 : p(x_1) \leftarrow p(x_1)$, and T_{nop} generates

$$C_2 : p(x_1) \leftarrow q(x_1), \sim r(x_1).$$

Let $eq/2$ be a unique predicate not occurring in $P_{nop} \cup P_{op}$, which is defined by a single clause $eq(x,x) \leftarrow$ (it can be implemented by the $=/2$–builtin for unification, which is generally provided by existing PROLOG-systems [SteSha86]). Then the negative non-operational literal $\sim r(x_1)$ can be eliminated by rewriting C_2 (after the necessary clause renaming) as

$$C_3 : p(x_1) \leftarrow q(x_1), \sim (eq(r(x_1), r(x_2)), v(x_2), \sim w(x_2)),$$
$$\sim (eq(r(x_1), r(x_3)), s(x_3, y_3), \sim t(y_3), \sim u(y_3)).$$

This step corresponds essentially to the reduction proposed by Siqueira and Puget [SiPu88], though it was described for definite programs.

As clause C_3 shows, $T_{\sim nop}$ introduces negated normal conjunctions as subgoals, which give rise to the following definition:

Definition 2 (normal$^+$) *A* normal$^+$-literal *is either an atom, or a term of the form* $\sim (L_1, \ldots, L_n)$*, where* $L_1, \ldots, L_n$ *are literals. A* normal$^+$-clause *is of the form* $A \leftarrow L_1, \ldots, L_n$*, where* A *is an atom and* $L_1, \ldots, L_n$ *are normal$^+$-literals. A* normal$^+$-goal *is of the form* $\leftarrow L_1, \ldots, L_n$*, where* $L_1, \ldots, L_n$ *are normal$^+$-literals. A* normal$^+$-program *is a set of normal$^+$-clauses.*

Thus, we have to generalize the notion of a *refutation* of $P \cup \{G\}$ for normal$^+$-programs P and normal$^+$-goals G. If the selected literal is positive, the derivation step can proceed in the same way as in an SLDNF-refutation. Negative literals selected in SLDNF-refutations must be ground. However, note that transformed clauses such as C_3 can only be used in a refutation if we weaken this restriction: The variables x_2, x_3, and y_3 are not *bindable* in the body of C_3, i.e. they do not occur in a positive literal, and hence can never be made ground. At least, we can forbid the selection of a negative subgoal if it could be further instantiated by first selecting different positive literals with common variables.

Definition 3 (SLDNF$^+$-refutation) *Let P be a normal$^+$-program, G be a normal$^+$-goal. An* SLDNF$^+$-refutation for $P \cup \{G\}$ *consists of a sequence $G_0 = G, G_1, \ldots G_n = \Box$ of normal$^+$-goals, a sequence $C_1, \ldots, C_n$ of variants of program clauses of P or negative normal$^+$-literals, and a sequence $\theta_1, \ldots, \theta_n$ of mgu's, such that for each $i \in \{1, \ldots, n-1\}$, if G_i has the form $\leftarrow L_1, \ldots, L_m$ and L_k (for some $k \in \{1, \ldots, m\}$) is the literal selected in G_i, then one of the following conditions is true:*

- *C_{i+1} is a variant of a clause in P and has the form $A \leftarrow A_1, \ldots, A_l$, θ_{i+1} is an mgu for L_k and A, and G_{i+1} is $\leftarrow (L_1, \ldots, L_{k-1}, A_1, \ldots, A_l, L_{k+1}, \ldots, L_m)\theta_{i+1}$.*
- *L_k is a negative subgoal $\sim A_k$, no variable in A_k is bindable in $L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$, and there is a finitely failed SLDNF-tree for $P \cup \{\leftarrow A_k\}$; in this case, θ_{i+1} is the identity substitution, C_{i+1} is L_k, and G_{i+1} is $\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_m$.*

An SLDNF-refutation is also an SLDNF$^+$-refutation, but not vice versa. Definition 1 and transformation 1 can be generalized in the straightforward way by replacing the words "SLDNF-refutation" and "normal" by the words "SLDNF$^+$-refutation" and "normal$^+$", respectively, and by adding a clause $eq(x, x) \leftarrow$ to P_{in}. Now we are in a position to define Siqueira and Puget's transformation in the following way.

Transformation 2 ($T_{\sim nop}$) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op} \cup \{eq(x, x) \leftarrow\}$, C be a normal$^+$-clause of the form $H \leftarrow M_1, \ldots, M_m$, where for some $k \in \{1, \ldots, m\}$, M_k has the form $\sim(L_1, \ldots, L_n)$. Suppose that R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_m\}$ with a sequence of goals $G_0, \ldots, G_r = \Box$, input clauses*

$Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$; let G_l ($l \in \{0, \ldots, r-1\}$) be the goal in which (an instance of) M_k is selected, and θ be the composition $\theta_1 \ldots \theta_l$.

A clause C^T is the result of transformation $T_{\sim nop}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *L_s (for some $s \in \{1, \ldots, n\}$) is an atom whose predicate symbol is defined in P_{nop}, and the clauses $H_i \leftarrow B_i$, for $i \in \{1, \ldots, q\}$, are variants of the defining clauses which have no variables in common with C.*
- *$L_s\theta$ is the selected literal in $\leftarrow (L_1, \ldots, L_n)\theta$.*
- *for each $i \in \{1, \ldots, q\}$, N_i is the term*
  $$\sim (L_1, \ldots, L_{s-1}, eq(L_s, H_i), B_i, L_{s+1}, \ldots, L_n).$$
- *C^T is $H \leftarrow M_1, \ldots, M_{k-1}, N_1, \ldots, N_q, M_{k+1}, \ldots, M_m$.*

Note that transformation $T_{\sim nop}$ depends on two choices made by the computation rule used in R: one for the complex subgoal M_k, and one for the literal L_s selected within M_k. It is essential that we allow for *arbitrary* computation rules, and do not confine attention to the standard PROLOG left-to-right evaluation: If the computation rule used in the example uncovers the failure of the subgoal by constructing a *small* SLDNF-tree, then this *efficiency* can be translated into efficiency of the derived clause. That is true in the sense that an execution of the new clause under a simple computation rule (e.g. that of PROLOG) mimics a computation using the original clause under a more suitable proof strategy. A detailed account of this idea is given in [BruSch89].

However, this dependency of the inference procedure is not considered in [SiPu88]. Instead, a post-processing step is proposed in this work in order to reduce the complexity of the obtained clause. Although we will not include it in our system, it is instructive to formulate it as a transformation rule.

Definition 4 (T_{simp}) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op} \cup \{eq(x,x) \leftarrow\}$, C be a normal$^+$-clause of the form $H \leftarrow M_1, \ldots, M_m$, where for some $k \in \{1, \ldots, m\}$, M_k has the form $\sim (L_1, \ldots, L_n)$. Suppose that R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_m\}$ with a sequence of goals $G_0, \ldots, G_r = \Box$, input clauses $Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$; let G_l ($l \in \{0, \ldots, r-1\}$) be the goal in which (an instance of) M_k is selected, and θ be the composition $\theta_1 \ldots \theta_l$.*

A clause C^T is the result of transformation $T_{simp}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *There is a subset of indices $\{i_1, \ldots, i_q\} \subset \{1, \ldots, n\}$ such that a finitely failed SLDNF-tree exists for $P_{in} \cup \{\leftarrow (L_{i_1}, \ldots, L_{i_q})\theta\}$.*
- *M'_k is $\sim(L_{i_1}, \ldots, L_{i_q})$, and C^T is*
$$H \leftarrow M_1, \ldots, M_{k-1}, M'_k, M_{k+1}, \ldots, M_m.$$

Ideally, it would be desirable to find a *minimal* such subset. However, the computation may become too expensive in the general case. Therefore, in [SiPu88] a greedy right-to-left strategy was applied, which is appropriate for PROLOG-refutations. Urbano improved this procedure and corrected a slight bug [Ur91].

3.3 Reducing explicit unifications within negated subgoals

Part of the performance gain achieved by EBG is due to the evaluation of unifications before the new rule is actually used at runtime. Intuitively, the more a negative subgoal is instantiated, the greater is the chance that its failure can be rapidly revealed, since one of its positive literals is not unifiable with the head of any program clause. This idea will be made precise in lemma 3 (section 4.3). Reduction of the literals of predicate *eq*/2 introduced by $T_{\sim nop}$ has not been addressed in [SiPu88], and is performed by the next transformation.

Transformation 3 (T_{eq}) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op} \cup \{eq(x, x) \leftarrow\}$, C be a normal$^+$-clause of the form $H \leftarrow M_1, \ldots, M_m$, where for some $k \in \{1, \ldots, m\}$, M_k has the form $\sim(L_1, \ldots, L_n)$. Suppose that R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_m\}$ with a sequence of goals $G_0, \ldots, G_r = \Box$, input clauses $Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$; let G_l ($l \in \{0, \ldots, r-1\}$) be the goal in which (an instance of) M_k is selected, and θ be the composition $\theta_1 \ldots \theta_l$.*

A clause C^T is the result of transformation $T_{eq}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *There is an $s \in \{1, \ldots, n\}$ such that L_s has the form $eq(t_1, t_2)$, for some terms t_1, t_2.*

- *one of the following cases applies:*

   a) *t_1 and t_2 are not unifiable, and C^T is*
   $$H \leftarrow M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m.$$
   b) *$n > 1$, $t_1\theta$ and $t_2\theta$ are not unifiable, and C^T is*
   $$H \leftarrow M_1, \ldots, M_{k-1}, \sim eq(t_1, t_2), M_{k+1}, \ldots, M_m.$$
   c) *$t_1\theta$ and $t_2\theta$ are unifiable; π is a renaming substitution for all variables in L_s which are not bindable in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$, with $range(\pi) \cap var(C) = \emptyset$; σ is an mgu for $t_1\pi$ and $t_2\pi$, M'_k is $\sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_k)\pi$, and C^T is*
   $$(H \leftarrow M_1, \ldots, M_{k-1}, M'_k, M_{k+1}, \ldots, M_m)\sigma.$$

Case a) removes a tautological subgoal. In case b), all the literals in the subgoal except the unification check are removed; hence it can be viewed as a special case of T_{simp} with a singleton subset of literals. Case c) applies to the second subgoal in C_3, where we obtain with $\pi = \{x_2/x_4\}$ and $\sigma = \{x_4/x_1\}$
$$C_4 : p(x_1) \leftarrow q(x_1), \sim(v(x_1), \sim w(x_1)),$$
$$\sim(eq(r(x_1), r(x_3)), s(x_3, y_3), \sim t(y_3), \sim u(y_3)),$$
and by another corresponding step
$$C_5 : p(x_1) \leftarrow q(x_1), \sim(v(x_1), \sim w(x_1)), \sim(s(x_1, y_3)), \sim t(y_3), \sim u(y_3)).$$
The renaming π is necessary to avoid side effects for different subgoals in the clause; this will become clear in the course of the example.

3.4 Reducing negative literals within negated subgoals

The next transformation removes nested negations:

Transformation 4 ($T_{\sim\sim}$) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op} \cup \{eq(x, x) \leftarrow\}$, C be a normal$^+$-clause of the form $H \leftarrow M_1, \ldots, M_m$, where for some $k \in \{1, \ldots, m\}$, M_k has the form $\sim(L_1, \ldots, L_n)$. Suppose that R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_m\}$ with a sequence of goals $G_0, \ldots, G_r = \Box$, input clauses $Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$; let G_l ($l \in \{0, \ldots, r-1\}$) be the goal in which (an instance of) M_k is selected, and θ be the composition $\theta_1 \ldots \theta_l$.*

A clause C^T is the result of transformation $T_{\sim\sim}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *L_s is a negative subgoal $\sim A_s$, for some $s \in \{1, \ldots, n\}$.*
- *$L_s\theta$ is the selected literal in $\leftarrow (L_1, \ldots, L_n)\theta$.*
- *one of the following cases applies:*

  a) *$P_{in} \cup \{\leftarrow A_s\theta\}$ has a finitely failed SLDNF-tree, and C^T is*
  $H \leftarrow M_1, \ldots, M_{k-1}, \sim (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_k), M_{k+1}, \ldots, M_m.$

  b) *there is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow A_s\theta\}$, and C^T is*
  $$H \leftarrow M_1, \ldots, M_{k-1}, A_s, M_{k+1}, \ldots, M_m.$$

Case a) in this definition is another special of T_{simp}. Case b) simplifies the negative subgoal to an atom, as in the case of $C_5 = H_5 \leftarrow B_5$, with M_k being the second subgoal: There is a refutation R_5 for $P_{in} \cup \{\leftarrow B_5\}$ which selects $\sim (v(a), \sim w(a))$ in the second step, and $P_{in} \cup \{\leftarrow v(a), \sim w(a)\}$ has a finitely failed SLDNF-tree in which $\sim w(a)$ is selected at the root. The resulting clause is

$$C_6 : p(x_1) \leftarrow q(x_1), w(x_1), \sim (s(x_1, y_3), \sim t(y_3), \sim u(y_3)).$$

3.5 Reducing operational literals within negated subgoals

Evidently, negative literals within subgoals whose predicate symbols are defined in P_{nop} ought to be removed by some transformation. But a problem arises if such a literal is not selected at the root of the respective finitely failed SLDNF-tree, as required by $T_{\sim\sim}$:

Suppose an attempt is made to apply the same kind of transformation as in $T_{\sim\sim}$ to the literal $\sim u(y_3)$ in C_6. According to the first case, the derived clause would be $p(x_1) \leftarrow q(x_1), w(x_1), \sim (s(x_1, y_3), \sim t(y_3))$. But then no refutation exists which uses this clause, since there is an SLDNF-refutation for $P_{in} \cup \{\leftarrow s(a, y_3), \sim t(y_3)\}$ with computed answer $\{y_3/c\}$.

On the other hand, the second case of $T_{\sim\sim}$ would yield a rule $C_7' : p(x_1) \leftarrow q(x_1), w(x_1), u(y_3)$, which is overly general and therefore incorrect: Suppose we use this clause together with another example program $P_{op}' = \{q(a) \leftarrow, w(a) \leftarrow, s(a, d) \leftarrow\}$. Then there is a refutation for $\{C_7'\} \cup P_{op}' \cup \{\leftarrow p(a)\}$, but this goal cannot be proven using the initial domain theory: $P_{nop} \cup P_{op}' \cup \{\leftarrow p(a)\}$ finitely fails, since $P_{nop} \cup P_{op}' \cup \{\leftarrow r(a)\}$ has an SLDNF-refutation.

We can solve this problem by introducing an additional transformation $T_{\sim op}$ for the case that an operational atom is selected within a negative subgoal. Roughly speaking, the transformed clause shall express that this operational goal has the same number of solutions as in the example, each of which leads to a failure of the respective derived goal.

This idea can be formulated by means of a unique predicate symbol *definition*/2 not occurring anywhere else. We can pretend that all clauses in P_{op} are facts (otherwise an auxiliary "black box" computation yields the results as "lemmas"). We augment the program P_{in} by *def*(P_{op}), according to the following definition:

Definition 5 *Let P be a set of facts. Then def(P) is the smallest set of clauses which contains for each predicate p in P a clause* $\text{definition}(p, [H_1, \ldots, H_n]) \leftarrow$*, where* $\{H_1 \leftarrow, \ldots, H_n \leftarrow\}$ *is the subset of clauses in P which have p in their head.*

In our example, *def*(P_{op}) is

$$\{\textit{definition}(q/1, [q(a)]) \leftarrow, \textit{definition}(v/1, [v(a)]) \leftarrow,$$
$$\textit{definition}(w/1, [w(a)]) \leftarrow, \textit{definition}(s/2, [s(a,b), s(a,c)]) \leftarrow\}.$$

Most existing PROLOG-systems offer a predicate *findall*/2, which enumerates by backtracking all proofs for a goal and returns a list of its respective instantiations in the order of occurrence [SteSha86]. For an n-ary predicate symbol q, a literal $\textit{definition}(q/n, [t_1, \ldots, t_m])$ can be implemented as $\textit{findall}(q(x_1, \ldots, x_n), [t_1, \ldots, t_m])$, where the x_i are distinct new variables. Definition 1 and transformations 1-4 can be generalized in the straightforward way by adding *def*(P_{op}) to P_{in}. Now we are in a position to define the last transformation operation:

Transformation 5 ($T_{\sim op}$) *Let P_{nop} and P_{op} be normal programs, P_{in} be the normal program $P_{nop} \cup P_{op} \cup \text{def}(P_{op}) \cup \{eq(x,x) \leftarrow\}$, C be a normal$^+$-clause of the form $H \leftarrow M_1, \ldots, M_m$, where for some $k \in \{1, \ldots, m\}$, M_k has the form $\sim(L_1, \ldots, L_n)$. Suppose that R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_m\}$ with a sequence of goals $G_0, \ldots, G_r = \Box$, input clauses $Q_1, \ldots, Q_r$, and mgu's $\theta_1, \ldots, \theta_r$; let G_l ($l \in \{0, \ldots, r-1\}$) be the goal in which (an instance of) M_k is selected, and θ be the composition $\theta_1 \ldots \theta_l$.*

A clause C^T is the result of transformation $T_{\sim op}(C, P_{nop}, P_{op}, R)$ if the following conditions hold:

- *There is an $s \in \{1, \dots, n\}$ such that L_s is an atom which is not defined in P_{nop}.*
- *$L_s\theta$ is the selected literal in $\leftarrow (L_1, \dots, L_n)\theta$.*
- *one of the following cases applies:*

  a) *$n > 1$, $L_s\theta$ is not unifiable with the head of a clause in P_{op}, and C^T is* $H \leftarrow M_1, \dots, M_{k-1}, \sim L_s, M_{k+1}, \dots, M_m$.

  b) *$L_s\theta$ is a ground term unifiable with the head of a clause in P_{op}, M'_k is $\sim (L_1, \dots, L_{s-1}, L_{s+1}, \dots, L_n)$, and C^T is*
  $$H \leftarrow M_1, \dots, M_{k-1}, M'_k, M_{k+1}, \dots, M_m.$$

  c) $\mathrm{def}(P_{op})$ *contains a clause* $\mathrm{definition}(p, [H_1, \dots, H_q])$*, where p is the predicate symbol of L_s; for each $i \in \{1, \dots, q\}$, N_i is the term $\sim (L_1, \dots, L_{s-1}, eq(L_s, x_i), L_{s+1}, \dots, L_n)$, where $x_1, \dots x_q$ are distinct variables not occurring in C; C^T is*
  $H \leftarrow M_1, \dots, M_{k-1}, \mathrm{definition}(p, [x_1, \dots, x_q]), N_1, \dots, N_q, M_{k+1}, \dots, M_m$.

Once again, a) and b) are special cases of T_{simp}. In b), the test of the selected literal may be discarded, since it cannot generate new bindings. Case c) applies to clause C_6 in the example, with M_k being the third subgoal. There is a refutation R_6 which selects it in the third goal, and constructs for it a finitely failed SLDNF-tree with the literal $L_s\theta = s(a, y_3)$ being selected at the root. Since there are two solutions in the example, we obtain

$$\begin{aligned} C_7 : p(x_1) \leftarrow & \, q(x_1), w(x_1), \mathit{definition}(s/2, [z_1, z_2]), \\ & \sim (\mathrm{eq}(s(x_1, y_3), z_1), \sim t(y_3), \sim u(y_3)), \\ & \sim (\mathrm{eq}(s(x_1, y_3), z_2), \sim t(y_3), \sim u(y_3)). \end{aligned}$$

It is easy to see that $P_{in} \cup \{\leftarrow B_7\}$ now has an SLDNF$^+$-refutation, if we let P_{in} be the program $P_{nop} \cup P_{op} \cup \{eq(x, x) \leftarrow\} \cup \mathit{def}(P_{op})$. Note that C_7 is more specific than C_6, since it can only be used in a refutation if P'_{op} contains *exactly two clauses* for $s/2$. Application of T_{eq} to the fourth subgoal of C_7 produces with $\pi = \{y_3/y_8\}$ and $\sigma = \{z_1/s(x_1, y_8)\}$

$$\begin{aligned} C_8 : p(x_1) \leftarrow & \, q(x_1), w(x_1), \mathit{definition}(s/2, [s(x_1, y_8), z_2]), \\ & \sim (\sim t(y_8), \sim u(y_8)), \sim (\mathrm{eq}(s(x_1, y_3), z_2), \sim t(y_3), \sim u(y_3)). \end{aligned}$$

Here the renaming π is necessary, since y_3 and y_8 in C_8 correspond to distinct constants a and b. Another application of T_{eq} yields

$$\begin{aligned} C_9 : p(x_1) \leftarrow & \, q(x_1), w(x_1), \mathit{definition}(s/2, [s(x_1, y_8), s(x_1, y_9)]), \\ & \sim (\sim t(y_8), \sim u(y_8)), \sim (\sim t(y_9), \sim u(y_9)). \end{aligned}$$

Now the literals $\sim t(y_8)$ and $\sim u(y_9)$ are selected at the roots of the respective finitely failed SLDNF-trees, and $T_{\sim\sim}$ can be used twice to obtain

$C_{11} : p(x_1) \leftarrow q(x_1), w(x_1), \mathit{definition}(s/2, [s(x_1, y_8), s(x_1, y_9)]), t(y_8), u(y_9).$

Two successive applications of T_{nop} to the literals $t(y_8)$ and $u(y_9)$ generate the final result

$$C_{13} : p(x_1) \leftarrow q(x_1), w(x_1), \mathit{definition}(s/2, [s(x_1, b), s(x_1, c)]).$$

4 Soundness of the transformation system

4.1 Requirements of EBG

Let us briefly review the general requirements of traditional EBG for definite programs, and transfer it to the case of normal programs considered here. Obviously, the derived clause is expected to be *operational*; it should be *sufficient* to explain that the target concept holds for the training example, and it has to be *correct* with respect to the initial domain theory when used together with different test examples. We formalize these issues in turn.

In traditional EBG, a clause is usually considered as operational if it describes the target concept in the same language in which the example is stated. Clearly, we want to maintain this idea, so the clause body may contain literals of predicate symbols p or atoms of the form $\mathit{definition}(p, t)$, where p is defined in P_{op}. Besides, we have to allow for negated unifications of the form $\sim eq(t_1, t_2)$ which result from transformation T_{eq}.

Furthermore, we expect the derived rule to have the same structure as the normal clauses in the domain theory to which it will be added; i.e. its body is supposed to consist exclusively of literals. This choice might possibly be controversial, since we could admit alternatively also complex (normal$^+$-) subgoals. However, it is not difficult to modify the operationality criterion in such a way that the transformation process stops at an earlier stage, as soon as all non-operational literals have been removed.

Definition 6 (Operationality) *Let C be a normal$^+$-clause $H \leftarrow L_1, \ldots, L_n$, P_{op} be a normal program in which the predicate symbols $eq/2$ and* definition/2 *do not occur. C is said to be* operational *wrt P_{op} if for each $i \in \{1, \ldots, n\}$, one of the following conditions hold:*

- *L_i is a literal whose predicate is defined in P_{op}.*

- *L_i has the form* definition(p, t)*, where p is a predicate symbol defined in P_{op}.*

- *L_i has the form $\sim eq(t_1, t_2)$, for some terms t_1, t_2.*

Next, suppose that there is an SLDNF$^+$-refutation for $P \cup \{G\}$ with computed answer σ; we define a normal$^+$-program P' to be *sufficient* for G and σ if we can derive a goal from it which is at least as general as $G\sigma$.

Definition 7 (Sufficiency) *Let P,P' be normal$^+$-programs, G be a normal$^+$-goal. Suppose there is an SLDNF-refutation for $P \cup \{G\}$ with computed answer σ. P' is said to be* sufficient *for G and σ if there is an SLDNF$^+$-refutation for $P' \cup \{G\}$ with computed answer σ', such that $G\sigma'\mu = G\sigma$, for some μ.*

A derived clause C should incorporate the entire knowledge from P_{nop} which is necessary for the proof that the training example is an instance of the target concept. If we have given an example refutation for $P_{in} \cup \{\leftarrow G\}$ with computed answer σ, we expect that $G\sigma$ can still be derived if P_{nop} is completely replaced by C. Thus, $P_{in} \setminus P_{nop} \cup \{C\}$ has to be sufficient for G and σ.

In the application phase of EBG, we replace the training example P_{op} by P'_{op} and update the respective clauses for *definition*/2 to obtain a program P_{test}. It has to be ensured that everything that can be derived from this new example by means of C could also have been derived by means of the original domain theory P_{nop}. Hence, $P_{test} \setminus P_{nop} \cup \{C\}$ ought to be *correct* wrt P_{test}, using the following definition:

Definition 8 (Correctness) *Let P,P' be normal$^+$-programs. P' is said to be* correct *wrt P if for each atomic goal G and each SLDNF$^+$-refutation for $P' \cup \{G\}$ with computed answer σ', there is also an SLDNF$^+$-refutation for $P \cup \{G\}$ with computed answer σ, such that $G\sigma\mu = G\sigma'$, for some μ.*

4.2 Overview

The following theorem states that the transformation system defined and illustrated in section 3 satisfies the requirements of operationality, sufficiency and correctness, as defined in the last section. It is true under the condition that (P_{nop}, P_{op}) is a so-called *EBG-admissible program pair* (definition 16); this restriction will be discussed in section 4.3.7.

Theorem 1 *Let (P_{nop}, P_{op}) be an EBG-admissible program pair, and A be an atom. Suppose that $P_{nop} \cup P_{op} \cup \{\leftarrow A\}$ has an SLDNF-refutation with computed answer σ. Then there is a transformation sequence for $(A\sigma, P_{nop}, P_{op})$ with a resulting clause C such that the following conditions hold:*

a) *C is operational wrt P_{op}.*

b) *$\{C\} \cup P_{op} \cup def(P_{op}) \cup \{eq(x,x) \leftarrow\}$ is sufficient for A and σ.*

c) *For each normal program P'_{op} such that (P_{nop}, P'_{op}) is an EBG-admissible program pair, $\{C\} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x) \leftarrow\}$ is correct wrt $P_{nop} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x) \leftarrow\}$.*

The proof is based on lemmas and preconditions which are summarized in section 4.3. Besides some general properties of SLDNF-trees stated in 4.3.1, these are generalizations of the *switching lemma* (section 4.3.3) and the *lifting lemma* (4.3.5) of [Ll87] which are designed to cover the selection of negative subgoals in SLDNF$^+$-refutations. The latter one cannot be established without further restrictions, since finite failure of an instance of a goal does not entail finite failure of the goal itself. Therefore, we impose on an EBG-admissible pair of a domain theory and a training example a concept of *allowedness* (section 4.3.4), which originally has been developed to avoid undesirable *floundering* computations. The fact that all intermediate clauses in the transformation sequence are *range-restricted* further facilitates the argument (section 4.3.6), since it permits to restrict attention to refutations of the clause body as goal and to ignore initial instantiations of the target concept.

The main part of the proof consists of two lines, corresponding to *sufficiency* (section 4.4) and *correctness* (section 4.5). They are both carried out in a series of lemmas, each of which deals separately with one transformation rule. The argument in section 4.4 is slightly more complex, since it also includes *operationality* of the last resulting clause, as well as *termination* of the transformation process.

It is shown that as long as the current transformed clause C_i is not operational wrt P_{op}, the preconditions of at least one transformation are satisfied for any refutation R_i, such that C_{i+1} and R_{i+1} can be constructed. Therefore, the sequence does not stop with a non-operational clause.

Termination is established by associating with SLDNF$^+$-refutations and finitely failed SLDNF-trees a so-called *complexity* which is non-increasing for successive refutations R_i and R_{i+1} in the sequence; if it is equal, then the transformation operation removes at least one literal from C_i. This well-founded ordering implies that the transformation sequence cannot be infinite. Finally, section 4.6 combines the results from sections 4.4 and 4.5 in the proof of theorem 1.

We assume the reader is familiar with the standard theoretical results of logic programming, which are contained in [Ll87]. The notation and terminology are consistent with [Ll87].

4.3 Basic definitions and lemmas

4.3.1 Finitely failed SLDNF-trees

The subsequent argument will be facilitated by stating once and for all some simple properties of finitely failed SLDNF-trees which will be used several times. If, for some goal G, $P \cup \{G\}$ has a finitely failed SLDNF-tree, then this is also true for a goal G' which is obtained from G by adding literals (lemma 1), as well as for an instance of this goal (lemma 2). Lemma 3 concerns goals G containing an atom L_s of the special literal $eq/2$ defined as above: $P \cup \{G\}$ has a finitely failed SLDNF-tree T if and only if it has a finitely failed SLDNF-tree T' in which L_s is selected at the root. Moreover, the size of T' cannot exceed the size of T, i.e. the set of its nodes can be mapped onto a subset of the nodes in T. This property will be used in the sufficiency argument for T_{eq} (lemma 16) to show that this transformation does not increase the complexity of the refutation for the respective clause body.

Lemma 1 *Let P be a normal program, $L_1, \ldots, L_n$ be literals. If, for some $s \in \{1, \ldots, n\}$, $P \cup \{\leftarrow L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n\}$ has a finitely failed SLDNF-tree, then there is also a finitely failed SLDNF-tree for $P \cup \{\leftarrow L_1, \ldots, L_n\}$.*

Proof: Obvious.

Lemma 2 (Subtree) *Let P be a normal program, G be a normal goal, θ be a substitution. If T is a finitely failed SLDNF-tree for $P \cup \{G\}$, then there is a finitely failed SLDNF-tree T' for $P \cup \{G\theta\}$, which is obtained by applying substitutions to the goals of a subtree of T and which has the property that, for any goal H in T which has a corresponding goal $H\phi$ in T', if L is selected in H, then the corresponding literal $L\phi$ is selected in $H\phi$.*

Proof: See [Ll91].

Lemma 3 *Let P be a normal program, and* eq/2 *be a predicate symbol defined by the single clause $eq(x,x)\leftarrow$ in P; let $L_1,\dots,L_n$ be literals, where for some $s \in \{1,\dots,n\}$, L_s has the form $eq(t_1,t_2)$, and t_1,t_2 are terms unifiable with mgu θ; let G be the goal $\leftarrow L_1,\dots,L_n$, and G' be $\leftarrow(L_1,\dots,L_{s-1},L_{s+1},\dots,L_n)\theta$.*
a) If $P \cup \{G'\}$ has a finitely failed SLDNF-tree, then this is also true for $P \cup \{G\}$.
b) If T is a finitely failed SLDNF-tree for $P\cup\{G\}$, then $P\cup\{G'\}$ has a finitely failed SLDNF-tree T', and there is a one-to-one correspondence which maps each goal H' in T' to a goal H in T such that, if L' is the literal which is selected in H', and L is the literal which is selected in H, then $L\phi = L'$, for some ϕ.

Proof: The first part of the claim is quite obvious: Assume that there is a finitely failed SLDNF-tree for $P\cup\{G'\}$. Then $\theta' := \{x'/t_1\theta\}\cup\theta$ is an mgu for L_s and the head of a variant $eq(x',x')\leftarrow$ of $eq(x,x)\leftarrow$. Since x' does not occur in $L_1,\dots,L_n$, $(L_1,\dots,L_{s-1},L_{s+1},\dots,L_n)\theta' = (L_1,\dots,L_{s-1},L_{s+1},\dots,L_n)\theta$. But there is no other clause in P for the predicate $eq/2$; hence this goal is the only child of the root in a finitely failed SLDNF-tree for $P \cup \{\leftarrow L_1,\dots,L_s,\dots,L_n\}$.

The second statement follows from an induction on the depth d of the tree. For $d = 0$, the literal L_k selected in G cannot be L_s. If L_k is an atom, then neither L_k nor its instance $L_k\theta$ are unifiable with the head of any clause in P. Otherwise, if L_k is a negative literal $\sim A$, then there is an SLDNF-refutation for $P \cup \{\leftarrow A\}$; since A is ground, the claim follows from $A\theta = A$.

Now suppose the result holds for finitely failed SLDNF-trees of depth at most $d-1$, and $P \cup \{G\}$ has a finitely failed SLDNF-tree of depth d. If L_s

is selected in G, then the claim follows immediately; therefore assume that a different literal L_k is selected, w.l.o.g. with $k < s$.

First suppose L_k is an atom, and $H \leftarrow B$ is a program clause in P such that L_k and H are unifiable with mgu σ; then there is a finitely failed SLDNF-tree T for $P \cup \{\leftarrow (L_1, \ldots, L_{k-1}, B, L_{k+1}, \ldots, L_n)\sigma\}$ of depth at most $d-1$. Either $L_k\theta$ does not unify with H, or there is an mgu σ' such that $\sigma\mu = \theta\sigma'$, for some substitution μ. According to lemma 2, $P \cup \{\leftarrow (L_1, \ldots, L_{k-1}, B, L_{k+1}, \ldots, L_n)\theta\sigma'\}$ has a finitely failed SLDNF-tree T', which is obtained by applying substitutions to the goals of a subtree of T and which has the property that, for any goal H in T which has a corresponding goal $H\phi$ in T', if L is selected in H, then the corresponding literal $L\phi$ is selected in $H\phi$. If an instance of $L_s\theta\sigma'$ is selected in some goal K in T', then its only child is the goal obtained from K by deleting this literal, since $t_1\theta\sigma' = t_2\theta\sigma'$. Therefore, by leaving out K and deleting (instances of) L_s from all the goals in T', a finitely failed SLDNF-tree T'' for $P \cup \{\leftarrow (L_1, \ldots, L_{k-1}, B, L_{k+1}, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta\sigma'\}$ can be constructed, where each goal H'' in T'' with selected literal L'' uniquely corresponds to a goal H in T with selected literal L such that $L\phi = L''$, for some ϕ. We can proceed in this way for all clauses in P to collect the trees rooted at the children of G' in a finitely failed SLDNF-tree for $P \cup \{G'\}$.

Now assume that L_k is a negative literal $\sim A$. Then there is a finitely failed SLDNF-tree T of depth $d-1$ for $P \cup \{\leftarrow L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_n\}$. According to the induction hypothesis, we obtain a finitely failed SLDNF-tree T' for $P \cup \{\leftarrow (L_1, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta\}$, where each goal H' in T' with selected literal L' uniquely corresponds to a goal H in T with selected literal L such that $L\phi = L'$, for some ϕ. Note that $A\theta = A$, since A is ground. Therefore, $P \cup \{G'\}$ has a finitely failed SLDNF-tree in which T' is rooted at the single child of G'. $\square$

4.3.2 Refutation complexity

The following definitions are intended to imply *termination* of the transformation process. A *complexity* number is associated with SLDNF$^+$-refutations and finitely failed SLDNF-trees. This is done in parallel and inductively on the rank (number of nested negations) of the refutation or SLDNF-tree. Roughly speaking, the *complexity* of a refutation (resp. of a finitely failed SLDNF-tree) is the number of goals in the main refutation (resp. the number of nodes in the tree) *plus* the complexity of auxiliary finitely failed SLDNF-trees (resp. SLDNF-refutations); goals whose selected literals have the special predicate symbols *eq*/2 or *definition*/2 are not counted.

Each SLDNF-refutation is of finite rank and length, and each finitely failed SLDNF-tree is also finite [Ll87]. Therefore, each SLDNF$^+$-refutation has a finite complexity.

As will be shown in section 4.4, these definitions are suitable to prove that a transformation sequence cannot be infinite, since the complexity is non-increasing for successive refutations in it.

Definition 9 *Let P be a normal program, G be a normal goal. The* complexity *$c(R)$ of an SLDNF-refutation R of rank* 0 *for $P \cup \{G\}$ with a sequence of goals $G_0 = G, G_1, \ldots, G_n = \Box$ is n.*

Definition 10 *Let P be a normal program, G be a normal goal, T be a finitely failed SLDNF-tree of rank* 0 *for $P \cup \{G\}$. We say T is of complexity $c(T)$ if $\{G_1, \ldots, G_n\}$ is the set of nodes (goals) in T, and $c(T)$ is $\Sigma_{i=1}^n c_i$, where for each $i \in \{1, \ldots, n\}$,*

$$c_i := \begin{cases} 0: & \text{the literal selected in } G_i \text{ has the predicate symbol eq/2} \\ 1: & \text{otherwise} \end{cases}$$

Definition 11 *Let P be a normal program, G be a normal goal, R be an SLDNF-refutation of rank $k+1$ for $P \cup \{G\}$ with the sequence of goals $G_0 = G, G_1, \ldots, G_n = \Box$. We say R is of complexity $c(R)$ if $c(R)$ is $\Sigma_{i=1}^n c_i$, where for each $i \in \{1, \ldots, n\}$,*

$$c_i := \begin{cases} c(T): & \text{a negative subgoal} \sim A \text{ is selected in } G_i, \text{ and } P \cup \{\leftarrow A\} \text{ has} \\ & \text{a finitely failed SLDNF-tree } T \text{ of rank } k \text{ and complexity } c(T) \\ 1: & \text{otherwise} \end{cases}$$

Definition 12 *Let P be a normal program, G be a normal goal, T be a finitely failed SLDNF-tree of rank $k+1$ for $P \cup \{G\}$. We say T is of complexity $c(T)$ if $\{G_1, \ldots, G_n\}$ is the set of nodes (goals) in T, and $c(T)$ is $\Sigma_{i=1}^n c_i$, where for each $i \in \{1, \ldots, n\}$,*

$$c_i := \begin{cases} 0: & \text{the literal selected in } G_i \text{ has the predicate symbol eq/2} \\ 1 + c(R): & G_i \text{ is a leaf node in which a negative literal} \sim A \text{ is selected,} \\ & \text{and } P \cup \{\leftarrow A\} \text{ has an SLDNF-refutation } R \text{ of rank } k \\ & \text{and complexity } c(R) \\ 1: & \text{otherwise} \end{cases}$$

Definition 13 *Let P be a normal program, G be a normal$^+$-goal, R be an SLDNF$^+$-refutation for $P \cup \{G\}$ with the sequence of goals $G_0 = G, G_1, \ldots, G_n = \Box$. We say R is of complexity $c(R)$ if $c(R)$ is $\Sigma_{i=1}^{n} c_i$, where for each $i \in \{1, \ldots, n\}$,*

$$c_i := \begin{cases} 0: & \textit{the literal selected in } G_i \textit{ has the predicate symbol } \mathrm{definition}/2 \\ c(T): & \textit{a negative subgoal} \sim A \textit{ is selected in } G_i\textit{, and there is a finitely} \\ & \textit{failed SLDNF-tree } T \textit{ for } P \cup \{\leftarrow A\} \textit{ of complexity } c(T) \\ 1: & \textit{otherwise} \end{cases}$$

4.3.3 Switching Lemma

Lemma 5 states that two literals L_i and L_j which are selected in subsequent goals in an SLDNF$^+$-refutation, can also be selected in reverse order without changing the computed answer, provided that L_j is positive. Furthermore, changing a refutation in this way can only reduce its complexity.

The proof is very similar to that in [Ll87]. Lemma 4 is used to release it from the involved technical detail concerning the following property of mgu's: Suppose we want to find an mgu for terms $[A_1, A_2]$ and $[B_1, B_2]$. Then we can start either with the first two literals A_1 and B_1, or with the second ones A_2 and B_2. However, this order is irrelevant for the resulting composed substitution (modulo variable renaming).

Lemma 4 (Reversibility of Mgu's) *Let A_1, A_2, B_1, B_2 be terms; suppose that θ_1 is an mgu for A_1 and B_1, and θ_2 is an mgu for $A_2\theta_1$ and $B_2\theta_1$. Then there are an mgu θ'_1 for A_2 and B_2 and an mgu θ'_2 for $A_1\theta'_1$ and $B_1\theta'_1$, such that $\theta_1\theta_2 = \theta'_1\theta'_2\rho'$ and $\theta'_1\theta'_2 = \theta_1\theta_2\rho$, for some substitutions ρ, ρ'.*

Proof: Since $A_2\theta_1\theta_2 = B_2\theta_1\theta_2$, we can unify A_2 and B_2. Let θ'_1 be an mgu of A_2 and B_2; then $\theta_1\theta_2 = \theta'_1\sigma$, for some substitution σ.

Furthermore, $A_1\theta'_1\sigma = A_1\theta_1\theta_2 = B_1\theta_1\theta_2 = B_1\theta'_1\sigma$. Hence we can unify $A_1\theta'_1$ and $B_1\theta'_1$. Suppose θ'_2 is an mgu. Thus $\sigma = \theta'_2\rho'$, for some ρ'. Consequently, $\theta_1\theta_2 = \theta'_1\theta'_2\rho'$.

Next, note that $A_1\theta'_1\theta'_2 = B_1\theta'_1\theta'_2$, but that θ_1 is an mgu for A_1 and B_1. Thus $\theta'_1\theta'_2 = \theta_1\gamma$, for some γ. But $A_2\theta_1\gamma = A_2\theta'_1\theta'_2 = B_2\theta'_1\theta'_2 = B_2\theta_1\gamma$. Thus γ unifies $A_2\theta_1$ and $B_2\theta_1$, and so $\gamma = \theta_2\rho$, for some ρ. Consequently, $\theta'_1\theta'_2 = \theta_1\theta_2\rho$. $\Box$

Lemma 5 (Switching) *Let P be a normal program, G be a normal$^+$-goal. Suppose that $P \cup \{G\}$ has an SLDNF$^+$-refutation R with the sequence of goals $G_0 = G, G_1, \ldots, G_n = \Box$, input clauses $C_1, \ldots, C_n$, and mgu's $\theta_1, \ldots, \theta_n$. Suppose G_{q-1} is $\leftarrow L_1, \ldots, L_i, \ldots, L_j, \ldots, L_k$, L_i is selected in G_{q-1}, and $L_j\theta_q$ is an atom selected in G_q.*

Then there exists an SLDNF$^+$-refutation R' for $P \cup \{G\}$, in which L_j is selected in G_{q-1} instead of L_i, and (the respective instance of) L_i is selected in G_q instead of L_j. If σ is the answer substitution computed by R, and σ' is the answer substitution computed by R', then $G\sigma$ and $G\sigma'$ are variants. Furthermore, R' has a complexity which is not greater than the complexity of R.

Proof: Suppose C_{q+1} has the form $C^+_{q+1} \leftarrow C^-_{q+1}$. If L_i is a negative subgoal $\sim A$, then $P \cup \{\leftarrow A\}$ has a finitely failed SLDNF-tree T, and θ_q is the empty substitution. Lemma 2 shows that there is a finitely failed SLDNF-tree T' for $P \cup \{\leftarrow A\theta_{q+1}\}$, whose complexity is lower than or equal to that of T. Therefore, the goals can be selected in reverse order without increasing the complexity, and the $(q+1)$-st goal in the new refutation is equal to G_{q+1}.

Now assume that L_i is an atom, and C_q has the form $C^+_q \leftarrow C^-_q$. Then G_q is $\leftarrow (L_1, \ldots, L_{i-1}, C^-_q, L_{i+1}, \ldots, L_j, \ldots, L_k)\theta_q$, and G_{q+1} is $\leftarrow (L_1, \ldots, L_{i-1}, C^-_q, L_{i+1}, \ldots, L_{j-1}, C^-_{q+1}, L_{j+1}, \ldots, L_k)\theta_q\theta_{q+1}$, where θ_q is an mgu for L_i and C^+_q, and θ_{q+1} is an mgu for $L_j\theta_q$ and $C^+_{q+1} = C^+_{q+1}\theta_q$. By lemma 4, there are an mgu θ'_q for L_j and C^+_{q+1} and an mgu θ'_{q+1} for $L_i\theta'_q$ and $C^+_q\theta'_q$, such that $\theta_q\theta_{q+1}$ and $\theta'_q\theta'_{q+1}$ are equal up to variable renaming. Clearly, we can assume that θ'_q does not act on any of the variables in C_q, so θ'_{q+1} is also an mgu for $L_i\theta'_q$ and C^+_q. Consequently, the literals can be selected in reverse order without affecting the complexity, and the $(q+1)$-st goal in the new refutation is a variant of G_{q+1}.

In either case, the remainder of the new refutation now proceeds in the same way as the given refutation (modulo variants), and the claim follows. $\Box$

4.3.4 Allowedness

In SLDNF-resolution, negative literals may only be selected if they are ground. This is an essential precondition to ensure correctness wrt the declarative concept of *completion* [Ll87]. However, it might happen that at some point a derivation *flounders*, i.e. only non-ground negative literals can be selected. Several concepts have been developed to exclude this undesirable behavior by imposing syntactical restrictions on the form of the programs and goals. We adopt one of them and state some useful implications in lemma 6; these will be necessary preconditions to generalize the lifting lemma of [Ll87] to the case that negative literals are selected in refutations, as will be explained in section 4.3.5.

Definition 14 (Allowedness) *Let C be a normal$^+$-clause $H \leftarrow L_1, \ldots, L_n$. We say C is* admissible *if every variable occurring in C occurs either in the head H or in a positive literal in $L_1, \ldots, L_n$. We say C is* allowed *if every variable occurring in C occurs in a positive literal in $L_1, \ldots, L_n$.*

Let P be a normal$^+$-program, G be a normal$^+$-goal. We say $P \cup \{G\}$ is allowed if the following conditions are satisfied:

- *Every clause in P is admissible.*
- *every clause in the definition of a predicate symbol occurring in a positive literal in the body of G or in a positive literal in the body of a clause in P is allowed.*

Lemma 6 *Let P be a normal program, G be a normal$^+$-goal. Suppose $P \cup \{G\}$ is allowed and has an SLDNF$^+$-refutation with the sequence of goals $G_0 = G, G_1, \ldots, G_n = \Box$, the sequence of input clauses $Q_1, \ldots, Q_n$, and the sequence of mgu's $\theta_1, \ldots, \theta_n$. Then the following conditions hold:*

i) $P \cup \{G_k\}$ is allowed, for each $k \in \{1, \ldots, n\}$.

ii) $\theta_1 \ldots \theta_n$ is a ground substitution for the bindable variables in G.

iii) If G contains a negative subgoal $\sim A$ which is selected in G_k (for some $k \in \{0, \ldots, n-1\}$), then $\theta_1 \ldots \theta_k$ is a ground substitution for all bindable variables in G occurring in A.

Proof: Suppose $P \cup \{G_i\}$ is allowed, for some $i \in \{0, \ldots, n-1\}$; let G_i have the form $\leftarrow L_1, \ldots, L_s, \ldots, L_p$, where L_s is the selected literal. If L_s is a negative subgoal, then the set of bindable variables in G_{i+1} is the same as in G_i. Otherwise, Q_{i+1} is a clause $H \leftarrow M_1, \ldots, M_m$, θ_{i+1} is an mgu for

L_s and H, and G_{i+1} is $\leftarrow(L_1,\ldots,L_{s-1},M_1,\ldots,M_m,L_{s+1},\ldots,L_p)\theta_{i+1}$. Since every clause in the definition of a predicate symbol occurring in a positive literal in $M_1,\ldots,M_m$ is allowed, $P\cup\{G_{i+1}\}$ must be allowed. Furthermore, Q_{i+1} is allowed, and thus all variables in $L_s\theta_{i+1}$ occur in a positive literal in $(M_1,\ldots,M_m)\theta_{i+1}$. Hence, for each variable x which occurs in a positive literal in G_i, all variables in $x\theta_{i+1}$ occur in a positive literal in G_{i+1}. Consequently, i) and ii) follow from a straightforward induction on the length of the refutation. iii) follows from the definition of an SLDNF$^+$-refutation (definition 3), since no variable occurring in $A\theta_1\ldots\theta_k$ may be bindable in G_k at the time it is selected. $\Box$

Lemma 6 can be used to strengthen the switching lemma 5 from section 4.3.3: Under the condition of allowedness, the computed answer substitutions must be equal.

Corollary 1 *Let P be a normal program, G be a normal$^+$-goal. Suppose that $P\cup\{G\}$ is allowed and has an SLDNF$^+$-refutation with the sequence of goals $G_0=G,G_1,\ldots,G_n=\Box$, input clauses $C_1,\ldots,C_n$, and mgu's $\theta_1,\ldots,\theta_n$. Suppose G_{q-1} is $\leftarrow L_1,\ldots,L_i,\ldots,L_j,\ldots,L_k$, L_i is selected in G_{q-1}, and $L_j\theta_q$ is an atom selected in G_q.*

Then there exists an SLDNF$^+$-refutation for $P\cup\{G\}$ with the same computed answer and of lower or equal complexity, in which L_j is selected in G_{q-1} instead of L_i, and (the respective instance of) L_i is selected in G_q instead of L_j.

Proof: Let σ be the answer substitution for G computed by the given refutation. It follows from the switching lemma 5 that a refutation for $P\cup\{G\}$ exists which selects L_i and L_j in reverse order, and whose complexity is not greater than that of the first one. If σ' denotes its computed answer, then $G\sigma$ and $G\sigma'$ are variants. Specifically, there must be a substitution μ such that $G\sigma\mu=G\sigma'$. Let x be a variable in G. If x does not occur in a positive literal in G, then it cannot be bound in any refutation, and $x\sigma=x\sigma'=x$. Otherwise, by lemma 6, $x\sigma$ is ground, and $x\sigma=x\sigma\mu=x\sigma'$. Consequently, $G\sigma=G\sigma'$. $\Box$

4.3.5 Lifting Lemma

The lifting lemma in [Ll87] states that for any *definite* program P, *definite* goal G, and substitution σ, the existence of an SLD-refutation for $P \cup \{G\sigma\}$ entails the existence of an SLD-refutation for $P \cup \{G\}$. Since we are referring to SLDNF$^+$-refutations, we have to impose some restrictions.

The first one is on the substitution σ, as the following example shows: Let P be $\{p(a) \leftarrow, r(a,b) \leftarrow\}$, and G be $\leftarrow p(x), \sim r(x,y)$. Then, with $\sigma := \{y/c\}$, $P \cup \{G\sigma\}$ has an SLDNF$^+$-refutation with computed answer $\{x/a\}$, but $P \cup \{G\}$ has none. The reason is that σ instantiates a variable which cannot be bound by any refutation. Thus, we have to insist that σ does only act on bindable variables in the goal.

The second restriction is on the program and the goal. Consider the program $P = \{p(x) \leftarrow, r(a) \leftarrow\}$, and the goal $G = \leftarrow p(y), \sim r(y)$. The substitution $\sigma := \{y/c\}$ satisfies the condition above, since it only instantiates the bindable variable y. Yet, there is no SLDNF$^+$-refutation for $P \cup \{G\}$, in contrast to the case of $P \cup \{G\sigma\}$. Obviously, finite failure of $P \cup \{\leftarrow A\phi\}$ does not imply failure of $P \cup \{\leftarrow A\}$.

This problem is circumvented if $P \cup \{G\}$ is allowed, since in this case the following property holds: Suppose there is a refutation R for $P \cup \{G\sigma\}$ which, in some goal, selects an instance $\sim A\sigma\theta$ of a negative goal $\sim A\sigma$. Then a refutation for $P \cup \{G\}$ can be constructed in which, by the time an instance $\sim A\theta'$ of $\sim A$ is selected, $A\theta'$ is a variant of $A\sigma\theta$. This favorable implication of allowedness was the main reason to impose it as a restriction on domain theory and training example.

Lemma 7 (Lifting) *Let P be a normal program, G be a normal$^+$-goal, σ be a substitution with the property that each variable occurring both in σ and in G is bindable in G. Suppose $P \cup \{G\sigma\}$ is allowed and has an SLDNF$^+$-refutation with a sequence of mgu's $\theta_1, \ldots, \theta_n$.*

Then there is also an SLDNF$^+$-refutation for $P \cup \{G\}$ with a sequence of mgu's $\theta'_1, \ldots, \theta'_n$ such that $\sigma\theta_1 \ldots \theta_n = \theta'_1 \ldots \theta'_n\mu$, for some substitution μ.

Proof: The proof is by induction on the length n of the refutation. First suppose that $n = 1$. Then G has the form $\leftarrow L_1$, $P \cup \{G\sigma\}$ has an SLDNF$^+$-refutation $G_0 = G\sigma, G_1 = \Box$ with input clause C_1 and mgu θ_1.

If L_1 is a negative subgoal $\sim A$, then $L_1\sigma = L_1$, because no variable in G occurs in a positive literal, and the claim follows immediately with $\theta'_1 = \theta_1$

and $\mu = \sigma$. If L_1 is an atom, then C_1 is a unit clause $H \leftarrow$, and θ_1 is an mgu for $L_1\sigma$ and H. We can assume that σ does not act on any of the variables of C_1; since $L_1\sigma\theta_1 = H\theta_1 = H\sigma\theta_1$, $\sigma\theta_1$ is a unifier for L_1 and H. Therefore, there must be an mgu θ_1' for L_1 and H such that $\sigma\theta_1 = \theta_1'\mu$, for some μ. Furthermore, $P \cup \{G\}$ has a refutation $G_0 = G, G_1 = \Box$ with input clause C_1 and mgu θ_1'.

Now suppose the result holds for refutations of length $n-1$; Assume G is $\leftarrow L_1, \ldots, L_m$, the subgoal selected in G is L_s, for some $s \in \{1, \ldots, m\}$, and $P \cup \{G\sigma\}$ has a refutation $G_0 = G, G_1, \ldots, G_n = \Box$ of length n with input clauses $C_1, \ldots, C_n$ and mgu's $\theta_1, \ldots, \theta_n$.

First suppose L_s is an atom, and C_1 has the form $H \leftarrow M_1, \ldots, M_k$. Clearly, we can assume that σ does not act on any of the variables of C_1. Since $L_s\sigma\theta_1 = H\theta_1 = H\sigma\theta_1$, $\sigma\theta_1$ is a unifier for L_s and H. Therefore, there must be an mgu θ_1' for L_s and H such that $\sigma\theta_1 = \theta_1'\rho$, for some ρ. Define $\rho' := \{x/t \in \rho | x \notin \text{domain}(\theta_1')\}$. We can assume that θ_1' is idempotent, and hence $\theta_1'\rho = \theta_1'\rho'$.

Let G_1' be the goal $\leftarrow (L_1, \ldots, L_{s-1}, M_1, \ldots, M_k, L_{s+1}, \ldots, L_m)\theta_1'$. First, in order to apply the induction hypothesis, it must be shown that ρ' does only act on bindable variables in G_1'. Let V be the set of variables in G not occurring in a positive literal. Since $\text{var}(\theta_1') \subseteq \text{var}(L_s) \cup \text{var}(H)$, it holds that $\text{var}(\theta_1') \cap V = \emptyset$. C_1 is allowed, hence V is also the set of variables in G_1' which do not occur in a positive literal. Furthermore, $\text{var}(\theta_1) \cap V = \emptyset$, since $\text{var}(\theta_1) \subseteq \text{var}(L_s\sigma) \cup \text{var}(H)$. Let x/t be a binding in ρ'. If t is not a variable, then x/t must be in $\sigma\theta_1$; if t is a variable y and $x/y \notin \sigma\theta_1$, then there must be a binding $y/x \in \theta_1'$. Hence, $\text{var}(\rho') \subseteq \text{var}(\sigma\theta_1) \cup \text{var}(\theta_1') \subseteq \text{var}(\sigma) \cup \text{var}(\theta_1) \cup \text{var}(\theta_1')$, and it follows that $\text{var}(\rho') \cap V = \emptyset$.

As $P \cup \{G_1'\rho'\}$ is allowed and has a refutation with the sequence of mgu's $\theta_2, \ldots, \theta_n$, the induction hypothesis yields a refutation for $P \cup \{G_1'\}$ with a sequence of mgu's $\theta_2', \ldots, \theta_n'$ such that $\rho'\theta_2 \ldots \theta_n = \theta_2' \ldots \theta_n'\mu$, for some μ. Hence, $\sigma\theta_1 \ldots \theta_n = \theta_1'\rho'\theta_2 \ldots \theta_n = \theta_1'\theta_2' \ldots \theta_n'\mu$.

Next suppose L_s is a negative subgoal $\sim A$; in this case, θ_1 is the empty substitution. Let G_1' be the goal $\leftarrow L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_m$. The set of non-bindable variables in G_1' is a subset of those in G, so they cannot occur in σ. According to the induction hypothesis, there is a refutation R for $P \cup \{G_1'\}$ with a sequence of mgu's $\theta_2', \ldots, \theta_n'$ such that $\sigma\theta_2 \ldots \theta_n = \theta_2' \ldots \theta_n'\mu$, for some μ. Let x be a variable occurring in A. If x is bindable in G_1', then it follows from lemma 6 that $x\theta_2' \ldots \theta_n'$ is ground, since $P \cup \{G_1'\}$ is allowed.

Consequently, $x\sigma\theta_2 \ldots \theta_n = x\theta'_2 \ldots \theta'_n\mu = x\theta'_2 \ldots \theta'_n$. Otherwise, $x\sigma = x$ is not bindable in $G'_1\sigma$, which is the first derived goal in the given refutation; hence, $x\sigma\theta_2 \ldots \theta_n = x = x\theta'_2 \ldots \theta'_n$. So it holds that $A\sigma\theta_2 \ldots \theta_n = A\theta'_2 \ldots \theta'_n$, and $P \cup \{\leftarrow A\theta'_2 \ldots \theta'_n\}$ has a finitely failed SLDNF-tree, according to lemma 2. We obtain an SLDNF$^+$-refutation for $P \cup \{G\}$ by adding one step at the end of R in which $L_s\theta'_2 \ldots \theta'_n$ is selected. Moreover, $\sigma\theta_1 \ldots \theta_n = \sigma\theta_2 \ldots \theta_n = \theta'_2 \ldots \theta'_n\mu$. □

If we only consider computed answers, then lemma 7 can be sharpened: the final instantiation of the goal in the new refutation is the the same as in the given one.

Corollary 2 *Let P be a normal program, G be a normal$^+$-goal, σ be a substitution with the property that each variable occurring both in σ and in G is bindable in G. Suppose $P \cup \{G\sigma\}$ is allowed and has an SLDNF$^+$-refutation with computed answer θ.*

Then there is also an SLDNF$^+$-refutation for $P \cup \{G\}$ of the same length with computed answer θ' such that $G\sigma\theta = G\theta'$.

Proof: It follows from the lifting lemma 7 that a refutation exists for $P \cup \{G\}$ of equal length and with computed answer θ' such that $G\sigma\theta = G\theta'\mu$, for some μ. Let x be a variable in G. If x is not bindable in G, then $x\sigma = x$ is not bindable in $G\sigma$, as well, and $x\sigma\theta = x\theta' = x$. Otherwise, by lemma 6, $x\theta'$ is ground, and so $x\theta' = x\theta'\mu = x\sigma\theta$. Consequently, $G\sigma\theta = G\theta'$. □

The next lemma is designed to be used in the correctness argument of T_{eq} (lemma 21). In case c) in the definition of this transformation, the clause body has the form $G, \sim(\text{eq}(M,N), R)$, and an mgu σ for M and N is applied to the whole body of the transformed clause, yielding $(G, \sim R)\sigma$. It has to be shown that this substitution is reversible: If a refutation exists for $P_{test} \cup \{\leftarrow G\sigma\}$ with a computed answer θ, and $P_{test} \cup \{\leftarrow R\sigma\theta\}$ has a finitely failed SLDNF-tree, then we can also construct a refutation for $P_{test} \cup \{\leftarrow G\}$ whose computed answer is θ', and afterwards unify the instances $M\theta'$ and $N\theta'$ with an mgu σ', such that $\leftarrow R\theta'\sigma'$ is also the root of a finitely failed SLDNF-tree, since it is a variant of $\leftarrow R\sigma\theta$.

Lemma 8 *Let P be a normal program, G be a normal$^+$-goal, T be a term of the form $eq(M,N), R$ with the properties that all variables occurring both in*

$[M,N]$ and in G are bindable in G, and M and N are unifiable with mgu σ. Suppose $P \cup \{G\sigma\}$ is allowed and has an SLDNF$^+$-refutation with a sequence of mgu's $\theta_1, \ldots, \theta_n$.

Then there is also an SLDNF$^+$-refutation for $P \cup \{G\}$ of the same length with a sequence of mgu's $\theta'_1, \ldots, \theta'_n$ such that $G\sigma\theta_1 \ldots \theta_n = G\theta'_1 \ldots \theta'_n$, and an mgu μ exists for $M\theta'_1 \ldots \theta'_n$ and $N\theta'_1 \ldots \theta'_n$ such that $T\sigma\theta_1 \ldots \theta_n$ is a variant of $T\theta'_1 \ldots \theta'_n\mu$.

Proof: The proof is by induction on the length n of the refutation. First suppose that $n = 1$; then G has the form $\leftarrow L_1$.

If L_1 is an atom, then the first input clause C_1 is a fact $H \leftarrow$, and θ_1 is an mgu for $L_1\sigma$ and $H = H\sigma$. It follows from lemma 4 that there are an mgu θ'_1 for L_1 and H and an mgu σ' for $M\theta'_1$ and $N\theta'_1$, such that $T\sigma\theta_1$ is a variant of $T\theta'_1\sigma'$. $P \cup \{G\}$ has a refutation $G_0 = G, G_1 = \Box$ with input clause C_1 and mgu θ'_1. Since C_1 is allowed, H must be ground; thus, $L_1\sigma\theta_1 = H\sigma\theta_1 = H = H\theta'_1 = L_1\theta'_1$.

If, on the other hand, L_1 is a negative subgoal, then no variable in G occurs in a positive literal, and consequently $\mathrm{var}([M,N]) \cap \mathrm{var}(G) = \emptyset$. We can assume that $\mathrm{var}(\sigma) \subseteq \mathrm{var}([M,N])$, and $\mathrm{var}(\theta_1) \subseteq \mathrm{var}(G)$. Consequently, $L_1\sigma = L_1$, and $[M,N]\theta_1 = [M,N]$; the claim follows immediately with $\theta'_1 = \theta_1$ and $\mu = \sigma$.

Now suppose the result holds for refutations of length $n-1$; assume G is $\leftarrow L_1, \ldots, L_m$, the subgoal selected in G is L_s, for some $s \in \{1, \ldots, m\}$, and $P \cup \{G\sigma\}$ has a refutation of length n with mgu's $\theta_1, \ldots, \theta_n$.

First assume that L_s is an atom. in this case, the first input clause C_1 has the form $H \leftarrow B_1, \ldots, B_k$. We can assume that σ does not act on any of the variables in C_1, so θ_1 is an mgu for $L_s\sigma$ and $H\sigma$. It follows from lemma 4 that there are an mgu θ'_1 for L_s and H and an mgu σ' for $M\theta'_1$ and $N\theta'_1$, such that $\sigma\theta_1$ and $\theta'_1\sigma'$ are equal up to variable renaming. If we denote the goal $\leftarrow L_1, \ldots, L_{s-1}, B_1, \ldots, B_k, L_{s+1}, \ldots, L_m$ as G_1, then there is an SLDNF$^+$-refutation for $P \cup \{G_1\theta'_1\sigma'\}$ with mgu's $\theta'_2, \ldots, \theta'_n$ such that $[G,T]\sigma\theta_1 \ldots \theta_n$ is a variant of $[G,T]\theta'_1\sigma'\theta'_2 \ldots \theta'_n$.

Let x be a variable in G. If x is bindable in G, then it follows from lemma 6 that $x\sigma\theta_1 \ldots \theta_n$ is ground, since $P \cup \{G\sigma\}$ is allowed. Consequently, $x\sigma\theta_1 \ldots \theta_n = x\theta'_1\sigma'\theta'_2 \ldots \theta'_n$. Therefore now assume that x is not bindable in G. Since $\mathrm{var}(\sigma) \subseteq \mathrm{var}([M,N])$, it holds that $x \notin \mathrm{var}(\sigma)$. Hence, $x\sigma = x$ is not bindable in $G\sigma$, and $x\sigma\theta_1 \ldots \theta_n = x$. On the other hand, $\mathrm{var}(\theta'_1) \subseteq \mathrm{var}([L_s, H])$, and $x\theta'_1 = x$ is not bindable in $G_1\theta'_1$. Moreover, $x \notin \mathrm{var}([M, N]\theta'_1)$, and with $\mathrm{var}(\sigma') \subseteq \mathrm{var}([M, N]\theta'_1)$, it follows that $x\theta'_1\sigma' = x$ is not bindable in $G_1\theta'_1\sigma'$. Thus, $x\theta'_1\sigma'\theta'_2 \ldots \theta'_n = x = x\sigma\theta_1 \ldots \theta_n$, and consequently $G\sigma\theta_1 \ldots \theta_n = G\theta'_1\sigma'\theta'_2 \ldots \theta'_n$.

All variables in $[M, N]\theta'_1$ which occur in $G_1\theta'_1$ are bindable in $G_1\theta'_1$, since C_1 is allowed. According to the induction hypothesis, there is a refutation for $P \cup \{G_1\theta'_1\}$ with mgu's $\theta''_2, \ldots, \theta''_n$, such that $G_1\theta'_1\sigma'\theta'_2 \ldots \theta'_n = G_1\theta'_1\theta''_2 \ldots \theta''_n$. Since $\mathrm{var}(G\theta'_1) \subseteq \mathrm{var}(G_1\theta'_1)$, also $G\theta'_1\sigma'\theta'_2 \ldots \theta'_n = G\theta'_1\theta''_2 \ldots \theta''_n$. Furthermore, an mgu μ exists for $M\theta'_1\theta''_2 \ldots \theta''_n$ and $N\theta'_1\theta''_2 \ldots \theta''_n$ such that $T\theta'_1\sigma'\theta'_2 \ldots \theta'_n$ is a variant of $T\theta'_1\theta''_2 \ldots \theta''_n\mu$.

Next suppose L_s is a negative subgoal $\sim A$; in this case, θ_1 is the empty substitution. Let G_1 be the goal $\leftarrow L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_m$; the set of non-bindable variables in G_1 is a subset of those in G, so they cannot occur in $[M, N]$. According to the induction hypothesis, there is a refutation R for $P \cup \{G_1\}$ with a sequence of mgu's $\theta'_2, \ldots, \theta'_n$ and an mgu μ for $M\theta'_2 \ldots \theta'_n$ and $N\theta'_2 \ldots \theta'_n$ such that $G_1\sigma\theta_2 \ldots \theta_n = G_1\theta'_2 \ldots \theta'_n$, and $T\sigma\theta_2 \ldots \theta_n$ is a variant of $T\theta'_2 \ldots \theta'_n\mu$. Let x be a variable occurring in A. If x is bindable in G_1, then it follows from lemma 6 that $x\theta'_2 \ldots \theta'_n$ is ground, since $P \cup \{G_1\}$ is allowed. Consequently, $x\sigma\theta_2 \ldots \theta_n = x\theta'_2 \ldots \theta'_n$. Otherwise, $x\sigma = x$ is not bindable in $G_1\sigma$, since $\mathrm{var}(\sigma) \subseteq \mathrm{var}([M, N])$; hence, $x\sigma\theta_2 \ldots \theta_n = x = x\theta'_2 \ldots \theta'_n$. So it holds that $A\sigma\theta_2 \ldots \theta_n = A\theta'_2 \ldots \theta'_n$, and $P \cup \{\leftarrow A\theta'_2 \ldots \theta'_n\}$ has a finitely failed SLDNF-tree, according to lemma 2. We obtain an SLDNF$^+$-refutation for $P \cup \{G\}$ by adding one step at the end of R, in which $L_s\theta'_2 \ldots \theta'_n$ is selected. $\square$

4.3.6 Range-restricted clauses

Suppose that for some program P and some atom G, $P \cup \{\leftarrow G\}$ has an SLDNF$^+$-refutation R which uses an input clause $C = H \leftarrow B$ in the first step. In the correctness argument for a clause C in the transformation sequence (section 4.5), we will have to consider the refutations for *all* such atoms G. However, this kind of proof can be facilitated for *range-restricted* clauses.

Definition 15 (Range-restricted clause) *A clause $H \leftarrow L_1, \ldots, L_n$ is called* range-restricted *if every variable occurring in the head H occurs in a positive literal in $L_1, \ldots, L_n$.*

If θ denotes an mgu for G and H, then the first derived goal in R is $\leftarrow B\theta$. The next two lemmas show that, if C is range-restricted, attention can be confined to refutations R' starting with the goal $\leftarrow B$, regardless of G; the initial instantiation θ can be ignored. Still, we have that the head H, instantiated with the answer substitution computed by R', yields the same atom as the computed answer of the original refutation R for $P \cup \{\leftarrow G\}$, applied to G. This idea is made precise by the following two lemmas.

Lemma 9 *Let P be a normal program and $C = H \leftarrow B$ be a range-restricted normal$^+$-clause, such that $P \cup \{\leftarrow B\}$ is allowed and has an SLDNF$^+$-refutation R with computed answer σ. Then, for each atom G that is unifiable with $H\sigma$, there is also an SLDNF$^+$-refutation R' for $P \cup \{C\} \cup \{\leftarrow G\}$ with computed answer σ' such that $G\sigma' = H\sigma$.*

Proof: Suppose $\theta_1, \ldots, \theta_n$ is the sequence of mgu's used in R. There must also be an SLDNF$^+$-refutation for $P \cup \{\leftarrow B\theta_1 \ldots \theta_n\}$ with a sequence of mgu's $\theta'_1, \ldots, \theta'_n$ such that $C\theta_1 \ldots \theta_n\theta'_1 \ldots \theta'_n = C\theta_1 \ldots \theta_n$. Let μ be an mgu for G and $H\theta_1 \ldots \theta_n$. As $\theta_1 \ldots \theta_n\mu$ is a unifier for G and H, there must be an mgu θ'' such that $\theta_1 \ldots \theta_n\mu = \theta''\rho$, for some ρ. Define $\rho' := \{x/t \in \rho | x \notin \text{domain}(\theta'')\}$. We can assume that θ'' is idempotent, and hence $\theta''\rho = \theta''\rho'$.

All variables in H are bindable in B, and it follows from lemma 6 that $H\theta_1 \ldots \theta_n$ is ground. Therefore $\text{var}(\mu) \subseteq \text{var}(G)$, and since we can assume that $\text{var}(G) \cap \text{var}(C) = \emptyset$, we have that $B\theta_1 \ldots \theta_n = B\theta_1 \ldots \theta_n\mu = B\theta''\rho$. First, in order to apply the lifting lemma, it must be shown that ρ' does only act on bindable variables in $B\theta''$. We have $\text{var}(\rho') \subseteq \text{var}(\theta_1 \ldots \theta_n\mu) \cup \text{var}(\theta'') \subseteq \text{var}(\theta_1 \ldots \theta_n) \cup \text{var}(G) \cup \text{var}(H)$. All variables occurring in H are bindable in B, as C is range-restricted; therefore, $var(\rho') \cap V = \emptyset$, where V denotes the set of variables in B not occurring in positive literals. But V is also the set of non-bindable variables in $B\theta''$; consequently, it follows that all variables occurring both in ρ' and $B\theta''$ are bindable in $B\theta''$.

Now corollary 2 to the lifting lemma yields an SLDNF$^+$-refutation for $P \cup \{\leftarrow B\theta''\}$ with a sequence of mgu's $\theta''_1, \ldots, \theta''_n$ such that $H\theta''\theta''_1 \ldots \theta''_n = H\theta''\rho'\theta'_1 \ldots \theta'_n = H\theta_1 \ldots \theta_n\mu\theta'_1 \ldots \theta'_n = H\theta_1 \ldots \theta_n$.

Let π be a renaming substitution for C, and θ'''_1 be an mgu for G and $H\pi$. Then $C\pi\theta'''_1$ is a variant of $C\theta''$, and an SLDNF$^+$-refutation exists for $P \cup \{C\} \cup \{\leftarrow G\}$ with a sequence of mgu's $\theta'''_1, \ldots, \theta'''_{n+1}$ such that $H\pi\theta'''_1 \ldots \theta'''_{n+1}$ is a variant of $H\theta''\theta''_1 \ldots \theta''_n$. By lemma 6, these goals are ground, as $P \cup$

$\{\leftarrow B\}$ is allowed and C is range-restricted. Consequently, $G\theta_1''' \ldots \theta_{n+1}''' = H\pi\theta_1''' \ldots \theta_{n+1}''' = H\theta''\theta_1'' \ldots \theta_n'' = H\theta_1 \ldots \theta_n$. $\Box$

Lemma 10 can be seen as the converse of lemma 9.

Lemma 10 *Let P be a normal program, $C = H \leftarrow B$ be a range-restricted normal$^+$-clause, G be an atom. Suppose $P \cup \{\leftarrow B\}$ is allowed, and there is an SLDNF$^+$-refutation R for $P \cup \{C\} \cup \{\leftarrow G\}$ in which exactly the first input clause is a variant of C. Then there is also an SLDNF$^+$-refutation R' for $P \cup \{\leftarrow B\}$; furthermore, if σ is the answer substitution for G computed by R, and σ' is the answer substitution for B computed by R', then $G\sigma = H\sigma'$.*

Proof: Let $\theta_1, \ldots, \theta_n$ be the sequence of mgu's used in R; suppose the first input clause in R is a variant $C\pi$ of C, such that $\mathrm{var}(G) \cap \mathrm{var}(C\pi) = \emptyset$. θ_1 is an mgu for $H\pi$ and G, and the second goal in R is $\leftarrow B\pi\theta_1$. Since C is range-restricted, all variables in $H\pi$ are bindable in $B\pi$. Therefore, as $\mathrm{var}(\theta_1) \subseteq \mathrm{var}(H\pi) \cup \mathrm{var}(G)$, we can apply the lifting lemma 7: There is an SLDNF$^+$-refutation for $P \cup \{\leftarrow B\pi\}$ with a sequence of mgu's $\theta_2', \ldots, \theta_n'$ such that $\theta_1 \ldots \theta_n = \theta_2' \ldots \theta_n'\mu$, for some μ. There must also be an SLDNF$^+$-refutation for $P \cup \{\leftarrow B\}$ with a sequence of mgu's $\theta_2'', \ldots, \theta_n''$ such that $H\theta_2'' \ldots \theta_n''$ is a variant of $H\pi\theta_2' \ldots \theta_n'$. But since $P \cup \{\leftarrow B\}$ is allowed and C is range-restricted, it follows from lemma 6 that both goals are ground, and $H\theta_2'' \ldots \theta_n'' = H\pi\theta_2' \ldots \theta_n' = H\pi\theta_2' \ldots \theta_n'\mu = H\pi\theta_1 \ldots \theta_n = G\theta_1 \ldots \theta_n$. $\Box$

The following example shows that lemma 10 does not hold for clauses which are not range-restricted: Let C be $p(x, y) \leftarrow q(x), \sim r(y)$, P be the program $\{q(a), r(c)\}$, G be $p(x, b)$. Then $P \cup \{C\} \cup \{\leftarrow G\}$ has an SLDNF$^+$-refutation with computed answer $\{x/a\}$, but $P \cup \{\leftarrow q(x), \sim r(y)\}$ has none.

4.3.7 Preconditions for domain theory and training example

Definition 16 summarizes the restrictions which will be imposed on the domain theory and the training example; they are intended to make the results from the preceding sections applicable in the soundness proof of the transformation system. Some have already been mentioned: The special predicate symbols *definition*/2 and $eq/2$ do not occur in the original programs; P_{op} does not contain predicate symbols defined in P_{nop}; it consists exclusively of facts. In addition to these conditions, all clauses in P_{nop} and P_{op} are required to be *allowed.* Although it might appears reasonable to exclude input programs which might lead to floundering computations, it has to be admitted that this notion is quite restrictive; a more liberal definition can e.g. be found in [Ll87].

Definition 16 (EBG-admissible program pair) *A pair* (P_{nop}, P_{op}) *is called an* EBG-admissible program pair *if* P_{nop} *and* P_{op} *are normal programs satisfying the following conditions:*

- *the predicate symbols* $eq/2$ *and* definition$/2$ *do not occur in* $P_{nop} \cup P_{op}$.
- *no predicate symbol occurring in* P_{op} *is defined in* P_{nop}.
- *all clauses in* $P_{nop} \cup P_{op}$ *are allowed.*
- *all clauses in* P_{op} *are facts.*

As a consequence of this definition, two important invariants hold throughout a transformation sequence: Each transformed clause C_i is range-restricted (lemma 11), and the input program (augmented by the definitions for the special predicates *eq*/2 and *definition*/2), together with the clause body as goal, satisfies the allowedness condition of definition 14 (corollary 3). The latter property follows immediately, since the only predicate of the input program defined by a clause that is not allowed is *eq*/2, which does not occur in a positive literal of the transformed clause body.

Definition 17 (EBG-admissible clause) *We say a normal*$^{+}$*-clause* $H \leftarrow B$ *is* EBG-admissible *if it is range-restricted, and no positive literal in* B *has the predicate symbol* eq/2.

Corollary 3 *Let* (P_{nop}, P_{op}) *be an EBG-admissible program pair,* C *be an EBG-admissible normal*$^{+}$*-clause* $H \leftarrow B$*,* P *be the normal program* $P_{nop} \cup P_{op} \cup def(P_{op}) \cup \{eq(x,x) \leftarrow\}$*. Then* $P \cup \{\leftarrow B\}$ *is allowed.*

Lemma 11 *Let* (P_{nop}, P_{op}) *be an EBG-admissible program pair,* C *be an EBG-admissible normal*$^{+}$*-clause,* R *be an SLDNF*$^{+}$*-refutation. Suppose* C^T *is the result of transformation* $T(C, P_{nop}, P_{op}, R)$*, where* T *is one of* T_{nop}*,* $T_{\sim nop}$*,* $T_{\sim op}$*,* $T_{\sim\sim}$*, or* T_{eq}*. Then* C^T *is EBG-admissible, too.*

Proof: Clearly, no transformation introduces a positive literal with predicate symbol $eq/2$. Suppose that C has the form $H \leftarrow M_1, \ldots, M_k, \ldots, M_n$, and the transformation is applied to literal M_k.

T_{nop} generates a clause $(H \leftarrow M_1, \ldots, M_{k-1}, A_1, \ldots, A_m, M_{k+1}, \ldots, M_n)\theta$, where $Q = A \leftarrow A_1, \ldots, A_m$ is a variant of a program clause in P_{nop}, and θ is an mgu for M_k and A. Since Q is allowed, all variables in $M_k\theta$ occur in a positive literal in $(A_1, \ldots, A_m)\theta$; Hence, also all variables in $H\theta$ occur in a positive literal in $(M_1, \ldots, M_{k-1}, A_1, \ldots, A_m, M_{k+1}, \ldots, M_n)\theta$, i.e. C^T is range-restricted.

In case c) of T_{eq}, C^T has the form $(H \leftarrow M_1, \ldots, M_{k-1}, M'_k, M_{k+1}, \ldots, M_n)\theta$, where both M_k and M'_k are negative subgoals. Since no positive literal is removed from the clause, all variables in $H\theta$ are bindable in the body.

Note that for all the remaining transformation rules, C^T has the form $H \leftarrow B^T$, where each positive literal in $M_1, \ldots, M_n$ is also contained in B^T. Therefore, the claim follows. □

This lemma completes the account of definitions and lemmas used in the soundness argument, and we can now turn to the main part of the proof. The next section treats the sufficiency of the transformation system.

4.4 Sufficiency

The first part of proposition 1 below states that for each clause C_i in the sequence which is not operational wrt P_{op} according to definition 6, the preconditions of at least one transformation are satisfied for any refutation R_i for its body, such that the sequence does not stop with a non-operational clause. The second part expresses that the transformed clause C_{i+1} can actually be applied to the example: From the refutation R_i used as input for this step, a refutation R_{i+1} can be constructed whose first goal is the body of C_{i+1}; moreover, the clause heads of C_i and C_{i+1}, instantiated with the answer substitutions computed by R_i and R_{i+1}, respectively, yield the same atom.

We use the definitions of section 4.3.2 to show that the complexity of R_{i+1} does not exceed that of R_i. In the case that it is equal, the transformation step removes at least one literal from C_i. Since the given example refutation terminates, it must be of finite complexity; therefore, this well-founded ordering implies that the transformation sequence cannot be infinite, as well.

The proof is carried out in a series of lemmas, each of which deals separately with one of the above-defined transformation rules.

Proposition 1 (Sufficiency) *Let (P_{nop}, P_{op}) be an EBG-admissible program pair, C be an EBG-admissible normal^{+}-clause $H \leftarrow B$, P_{in} be the normal program $P_{nop} \cup P_{op} \cup def(P_{op}) \cup \{eq(x,x) \leftarrow\}$. Suppose that there is an SLDNF^{+}-refutation R for $P_{in} \cup \{\leftarrow B\}$. Then the following conditions hold:*

a) *if C is not operational wrt P_{op}, then the conditions in at least one definition of a transformation $T(C, P_{nop}, P_{op}, R)$ are satisfied, where T is $T_{nop}, T_{\sim nop}, T_{\sim op}, T_{\sim\sim}$, or T_{eq}.*

b) *If this transformation exists, then its result is a clause $C^T = H^T \leftarrow B^T$, such that there is an SLDNF^{+}-refutation R^T for $P_{in} \cup \{\leftarrow B^T\}$ with the following properties:*

- *if σ is the answer substitution for B computed by R, and σ^T is the answer substitution for B^T computed by R^T, then $H\sigma = H^T\sigma^T$.*
- *if R has complexity $c(R)$, then R^T is of complexity $c(R^T)$ such that either $c(R^T) < c(R)$, or $c(R^T) = c(R)$ and the number of literals in C^T is lower than the number of literals in C.*

Proof: a) Suppose C is not operational wrt P_{op}. If
i) B contains an atom whose predicate is defined in P_{nop}, then T_{nop} is applicable. Otherwise, according to definition 6, B must contain a negative subgoal M of the form $\sim(L_1, \ldots, L_n)$, and there must be an $s \in \{1, \ldots, n\}$ such that one of the following conditions holds:
ii) L_s is an atom with predicate symbol $eq/2$, and $n > 1$; or L_s is the selected literal in (an instance of) $\leftarrow L_1, \ldots, L_n$, and is
iii) an atom whose predicate symbol is defined in P_{nop}, or
iv) an atom whose predicate symbol is defined in P_{op}, and $n > 1$; or
v) a negative subgoal.

In case ii) (resp. iii), iv) and v)), the preconditions of transformation T_{eq} (resp. $T_{\sim nop}$, $T_{\sim op}$, $T_{\sim\sim}$) are satisfied.

b) follows from lemmas 12 - 16 below. □

Lemma 12 *Proposition 1 b) holds for $T = T_{nop}$.*

Proof: Suppose C is a clause $H \leftarrow B$, where B has the form $L_1, \ldots, L_k$, $\ldots, L_m$, and R is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow B\}$ with a sequence of goals $G_0 = \leftarrow B$, $G_1, \ldots, G_r = \Box$, a sequence of input clauses $Q_1, \ldots, Q_r$, and a sequence of mgu's $\theta_1, \ldots \theta_r$. Assume G_l is the goal in which (an instance of) L_k is selected, and Q_{l+1} has the form $A \leftarrow A_1, \ldots, A_n$. By corollary 3, $P_{in} \cup \{\leftarrow B\}$ is allowed, and we can repeatedly apply corollary 1 to obtain an SLDNF$^+$-refutation R' for $P_{in} \cup \{\leftarrow B\}$ in which L_k is selected first, and whose complexity $c(R')$ not greater than the complexity $c(R)$ of R. If we denote the sequence of mgu's used in R' as $\theta, \theta'_2, \ldots, \theta'_r$, then $H^T\theta'_2 \ldots \theta'_r = H\theta\theta'_2 \ldots \theta'_r = H\theta_1 \ldots \theta_r$. The first derived goal in R' is $\leftarrow (L_1, \ldots, L_{k-1}, A_1, \ldots, A_n, L_{k+1}, \ldots, L_m)\theta$, which is equal to $\leftarrow B^T$, according to the definition of transformation T_{nop}. This new refutation R^T for $P_{in} \cup \{\leftarrow B^T\}$ leaves out one step performed in R', i.e. it has a complexity $c(R^T) = c(R') - 1 \leq c(R) - 1$. $\Box$

Lemma 13 *Proposition 1 b) holds for $T = T_{\sim nop}$.*

Proof: Let C be a clause $H \leftarrow B$, where B is $M_1, \ldots, M_k, \ldots, M_m$, and M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$. Suppose there is an SLDNF$^+$-refutation R for $P_{in} \cup \{\leftarrow B\}$ with a sequence of goals $G_0 = \leftarrow B, G_1, \ldots$, $G_r = \Box$, and a sequence of mgu's $\theta_1, \ldots \theta_r$. Let G_l be the goal in which (an instance of) M_k is selected, and assume $L_s\theta_1 \ldots \theta_l$ is the literal selected at the root of the finitely failed SLDNF-tree T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_n)\theta_1 \ldots \theta_l\}$.

Let $\{H_i \leftarrow B_i | i \in \{1, \ldots, q\}\}$ be a set of variants of the clauses in P_{nop} defining the predicate symbol in L_s which have no variables in common with C. If, for some $i \in \{1, \ldots, q\}$, $L_s\theta_1 \ldots \theta_l$ and H_i are not unifiable, then $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(L_s, H_i), B_i, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$ has a finitely failed SLDNF-tree T'_i which consists only of the root node; in this case, define $c_i := 0$.

Otherwise, if σ_i is the respective mgu, then there is a finitely failed SLDNF-subtree T_i of T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, B_i, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\sigma_i\}$; according to lemma 3, there is also a finitely failed SLDNF-tree T'_i for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(L_s, H_i), B_i, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$, with the same complexity $c_i := c(T'_i) = c(T_i)$.

For all $i \in \{1, \ldots, q\}$, let N_i be $\sim(L_1, \ldots, L_{s-1}, eq(L_s, H_i), B_i, L_{s+1}, \ldots, L_n)$. We obtain from R an SLDNF$^+$-refutation R^T for $P_{in} \cup \{\leftarrow B^T\} = P_{in} \cup$

$\{\leftarrow M_1, \ldots, M_{k-1}, N_1, \ldots, N_q, M_{k+1}, \ldots, M_m\}$ of length $r + q - 1$ which selects the (instances of the) literals $N_1, \ldots, N_q$ instead of selecting M_k in step l; the computed answer substitution for $H = H^T$ is the same as in R.

Note that according to definitions 10 and 12, T is of complexity $c(T) = 1 + \Sigma_{i=1}^{q} c_i$; but since for all $i \in \{1, \ldots, q\}$, $c(T_i') = c_i$, it follows that $c(R^T) = c(R) - c(T) + \Sigma_{i=1}^{q} c(T_i') = c(R) - 1$. $\square$

Lemma 14 *Proposition 1 b) holds for* $T = T_{\sim op}$.

Proof: Let C be a clause $H \leftarrow B$, where B is $M_1, \ldots, M_k, \ldots, M_m$, M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$, and L_s is a literal of a predicate symbol p which is not defined in P_{nop}. Suppose there is an SLDNF$^+$-refutation R for $P_{in} \cup \{\leftarrow B\}$ with a sequence of goals $G_0 = \leftarrow B, G_1, \ldots, G_r = \square$, and a sequence of mgu's $\theta_1, \ldots \theta_r$. Let G_l be the goal in which (an instance of) M_k is selected, and assume $L_s\theta_1 \ldots \theta_l$ is the literal selected at the root of the finitely failed SLDNF-tree T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_n)\theta_1 \ldots \theta_l\}$.

a) If $n > 1$ and $L_s\theta_1 \ldots \theta_l$ is not unifiable with the head of a clause in P_{op}, then a finitely failed SLDNF-tree T' exists for $P_{in} \cup \{\leftarrow L_s\theta_1 \ldots \theta_l\}$, and there is an SLDNF$^+$-refutation R^T for $P_{in} \cup \{\leftarrow M_1, \ldots, M_{k-1}, \sim L_s, M_{k+1}, \ldots, M_m\}$ $= P_{in} \cup \{\leftarrow B^T\}$ with the same sequence of mgu's as in R. Furthermore, R^T has the same complexity as R, since $c(T) = c(T') = 1$; in the transformed clause C^T, $n - 1 \geq 1$ literals are deleted from C.

b) Suppose $L_s\theta_1 \ldots \theta_l$ is a ground term which is unifiable with the head of a clause $Q \in P_{op}$. Hence the restriction of the respective mgu to the variables in the root node in T is the empty substitution, and its only child is the goal $\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l$. Therefore, we have an SLDNF$^+$-refutation R^T for $P_{in} \cup \{\leftarrow B^T\} = P_{in} \cup \{\leftarrow M_1, \ldots, M_{k-1}, \sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n), M_{k+1}, \ldots, M_m\}$ with the same sequence of mgu's as in R. Furthermore, $c(R^T) = c(R) - 1$, since the generation of the root node of T is left out in R^T.

c) Let the clauses for p in P_{op} be $H_1 \leftarrow, \ldots, H_q \leftarrow$, and $x_1, \ldots, x_q$ be distinct variables not occurring in C. Then there is a an SLDNF$^+$-refutation for def$(P_{op}) \cup \{\leftarrow \mathit{definition}(p, [x_1, \ldots, x_q])\}$ with an mgu $\theta' = \{x_i/H_i | i \in \{1, \ldots q\}\}$.

For all $i \in \{1, \ldots, q\}$, let N_i be $\sim(L_1, \ldots, L_{s-1}, eq(L_s, x_i), L_{s+1}, \ldots, L_n)$. Then we have that $N_i\theta_1 \ldots \theta_l\theta' = N_i\theta_1 \ldots \theta_l\{x_i/H_i\}$ (since $x_j \notin \text{var}(N_i)$, for $i \neq j$) $= \sim(L_1, \ldots, L_{s-1}, eq(L_s, H_i), L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l$, since $x_i \notin \text{var}(C)$.

If $L_s\theta_1 \ldots \theta_l$ and H_i are not unifiable, then there is a finitely failed SLDNF-tree T_i' for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(L_s, H_i), L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$ which consists only of the root node; in this case, define $c_i := 0$.

Otherwise, if σ_i is the respective mgu, then there is a finitely failed SLDNF-subtree T_i of T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\sigma_i\}$; according to lemma 3, there is also a finitely failed SLDNF-tree T_i' for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(L_s, H_i), L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$ of the same complexity $c_i := c(T_i') = c(T_i)$.

Since the variables in θ' do not occur in C, we can obtain from R an SLDNF$^+$-refutation R^T of length $r + q$ for $P_{in} \cup \{\leftarrow B^T\} = P_{in} \cup \{M_1, \ldots, M_{k-1}, \mathit{definition}(p, [x_1, \ldots, x_q]), N_1, \ldots, N_q, M_{k+1}, \ldots, M_m\}$, which selects the (instances of the) literals $\mathit{definition}(p, [x_1, \ldots, x_q]), N_1, \ldots, N_q$ instead of selecting M_k in step l; the computed answer substitution for $H = H^T$ is the same as in R.

It follows from definitions 10 and 12 that T is of complexity $c(T) = 1 + \Sigma_{i=1}^{q} c_i$; furthermore, according to definition 13, the selection of a literal with predicate symbol $\mathit{definition}/2$ does not increase the complexity of an SLDNF$^+$-refutation. Consequently, since for all $i \in \{1, \ldots, q\}$, $c(T_i') = c_i$, R^T is of complexity $c(R^T) = c(R) - c(T) + \Sigma_{i=1}^{q} c(T_i') = c(R) - 1$. $\square$

Lemma 15 *Proposition 1 b) holds for* $T = T_{\sim\sim}$.

Proof: Let C be a clause $H \leftarrow B$, where B is $M_1, \ldots, M_k, \ldots, M_m$, M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$, and L_s is a negative subgoal $\sim A_s$. Suppose there is an SLDNF$^+$-refutation R for $P_{in} \cup \{\leftarrow B\}$ with a sequence of goals $G_0 = \leftarrow B, G_1, \ldots, G_p = \square$, and a sequence of mgu's $\theta_1, \ldots \theta_p$. Let G_l be the goal in which (an instance of) M_k is selected, and assume $L_s\theta_1 \ldots \theta_l$ is the literal selected at the root of the finitely failed SLDNF-tree T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_n)\theta_1 \ldots \theta_l\}$. Then, according to the definition of a finitely failed SLDNF-tree, $L_s\theta_1 \ldots \theta_l$ is ground, and one of the following cases applies:

a) There are finitely failed SLDNF-trees both for $P_{in} \cup \{\leftarrow A_s\theta_1 \ldots \theta_l\}$ and for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$; call the latter one T'. In this case, it follows immediately that there is an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow M_1, \ldots, M_{k-1}, \sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n), M_{k+1}, \ldots, M_m\} = P_{in} \cup \{\leftarrow B^T\}$ with the same sequence of mgu's which constructs in step $l+1$ T' instead of T. According to definition 12, T has complexity $c(T) = c(T') + 1$; Consequently, R^T is of complexity $c(R^T) = c(R) - c(T) + c(T') = c(R) - 1$.

b) T consists of only the root node, and there is an SLDNF-refutation S' for $P_{in} \cup \{\leftarrow A_s\theta_1 \ldots \theta_l\}$. Denote the sequence of mgu's used in S' as $\theta'_1, \ldots, \theta'_q$. We can leave out step $l+1$ and append S' at the end of R to obtain an SLDNF$^+$-refutation R^T for $P_{in} \cup \{\leftarrow M_1, \ldots, M_{k-1}, L_s, M_{k+1}, \ldots, M_m\} = P_{in} \cup \{\leftarrow B^T\}$. Furthermore, $\mathrm{var}(\theta'_1 \ldots \theta'_q) \cap \mathrm{var}(C\theta_1 \ldots \theta_l) \subseteq \mathrm{var}(L_s\theta_1 \ldots \theta_l) = \emptyset$, since $L_s\theta_1 \ldots \theta_l$ is ground. Consequently, $H\theta_1 \ldots \theta_l\theta_{l+2} \ldots \theta_p\theta'_1 \ldots \theta'_q = H\theta_1 \ldots \theta_p$. According to definition 12, T is of complexity $c(T) = c(S') + 1$; therefore, $c(R^T) = c(R) - c(T) + c(S') = c(R) - 1$. □

Lemma 16 *Proposition 1 b) holds for* $T = T_{eq}$.

Proof: Let C be a clause $H \leftarrow B$, where B is $M_1, \ldots, M_k, \ldots, M_m$, M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$, and L_s has the form $eq(t_1, t_2)$. Suppose there is an SLDNF$^+$-refutation R for $P_{in} \cup \{\leftarrow B\}$ with a sequence of goals $G_0 = \leftarrow B, G_1, \ldots, G_p = \Box$, and a sequence of mgu's $\theta_1, \ldots \theta_p$; Let G_l be the goal in which (an instance of) M_k is selected.

If cases a) or b) of transformation 3 hold, the existence of R^T is obvious. In a), the finitely failed SLDNF-tree T for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_n)\theta_1 \ldots \theta_l\}$ consists only of the root node, and R^T differs from R only in that it omits step $l+1$, in which it is constructed. In b), it follows from lemma 1 that R^T constructs a finitely failed SLDNF-tree T' instead of T with the same complexity $c(T') = c(T) = 0$. Hence, in both these cases it holds that $c(R^T) = c(R)$.

Now assume that condition c) is true, i.e. $t_1\theta_1 \ldots \theta_l$ and $t_2\theta_1 \ldots \theta_l$ are unifiable. Let π be a renaming substitution for all variables in L_s which are not bindable in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$, with $\mathrm{range}(\pi) \cap \mathrm{var}(C) = \emptyset$. Since $M_k\pi\theta_1 \ldots \theta_l$ is a variant of $M_k\theta_1 \ldots \theta_l$, a finitely failed SLDNF-tree exists for $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(t_1, t_2), L_{s+1}, \ldots, L_n)\pi\theta_1 \ldots \theta_l\}$. Let ρ be an mgu for $t_1\pi\theta_1 \ldots \theta_l$ and $t_2\pi\theta_1 \ldots \theta_l$. According to lemma 3, $P_{in} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\pi\theta_1 \ldots \theta_l\rho\}$ has a finitely failed SLDNF-tree T', and there is a one-to-one correspondence which maps each goal H' in T' to a goal H in T such that, if L' is the literal which is selected in H', and L is the literal which is selected in H, then $L\phi = L'$, for some ϕ. Therefore, the complexity of T' is not greater than that of T.

Let M'_k be $\sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\pi$, and θ'_{l+1} be $\rho \cup \{x'/t_1\pi\theta_1 \ldots \theta_l\rho\}$. θ'_{l+1} is an mgu for $eq(t_1, t_2)\pi\theta_1 \ldots \theta_l$ and the head of a variant $eq(x', x') \leftarrow$ of the program clause $eq(x, x) \leftarrow$. Note that $M'_k\pi\theta_1 \ldots \theta_l\theta'_{l+1} = M'_k\pi\theta_1 \ldots \theta_l\rho$.

By corollary 3, $P_{in} \cup \{\leftarrow B\}$ is allowed, and it follows from lemma 6 that $\theta_1 \dots \theta_l$ is a ground substitution for all variables in $eq(t_1, t_2)\pi$ which are bindable in B. Hence $\mathrm{var}(eq(t_1, t_2)\pi\theta_1 \dots \theta_l) = \mathrm{range}(\pi)$, and these variables do not occur in C. Consequently, an SLDNF$^+$-refutation R' for $P_{in} \cup \{\leftarrow M_1, \dots, M_{k-1}, eq(t_1\pi, t_2\pi), M'_k, M_{k+1}, \dots, M_m\}$ can be constructed which proceeds in the same way as R, except that it selects in step $l+1$ instead of M_k the literal $eq(t_1\pi, t_2\pi)$, followed by M'_k. The sequence of mgu's used in this refutation is $\theta_1, \dots, \theta_l$, $\theta'_{l+1}, \theta_{l+1}, \dots \theta_p$, and $H\theta_1 \dots \theta_l\theta'_{l+1}\theta_{l+1} \dots \theta_p = H\theta_1 \dots \theta_l\theta_{l+1} \dots \theta_p$. By definition 13, $c(R') = c(R) - c(T) + c(T') + 1 \leq c(R) + 1$.

By corollary 3 and repeated application of corollary 1, we obtain from R' an SLDNF$^+$-refutation R'' for the same goal and of lower or equal complexity, in which $eq(t_1\pi, t_2\pi)$ is selected first. If $\theta''_1, \dots, \theta''_{p+1}$ denotes the sequence of mgu's used in R'', then $H\theta''_1 \dots \theta''_{p+1} = H\theta_1 \dots \theta_p$.

θ''_1 is an mgu for $eq(x', x')$ and $eq(t_1\pi, t_2\pi)$, i.e. $\theta''_1 = \sigma \cup \{x'/t_1\pi\sigma\}$, where σ is an mgu for $t_1\pi$ and $t_2\pi$. Therefore, the second goal in R'' is $\leftarrow(M_1, \dots, M_{k-1}, M'_k, M_{k+1}, \dots, M_m)\sigma$, which is equal to $\leftarrow B^T$, according to the definition of T_{eq}. The resulting SLDNF$^+$-refutation R^T for $P_{in} \cup \{\leftarrow B^T\}$ has the complexity $c(R^T) = c(R'') - 1 \leq c(R') - 1 \leq c(R)$. Since in all three cases of T_{eq} the number of literals in C^T is lower than the number of literals in C, the claim follows. □

4.5 Correctness

Suppose we have transformed a clause C_i into a clause C_{i+1} using domain theory P_{nop} and training example P_{op}. In the application phase of EBG, we replace the training example P_{op} by P'_{op} and update the respective clauses for *definition*/2 to obtain a program P_{test}. Then, if a refutation is given whose first goal is the body of the transformed clause C_{i+1}, proposition 2 states that we can also construct a refutation for the body of the initial clause C_i, such that both clause heads, instantiated with the respective answer substitutions, yield the same atom.

As in the previous section, the proof is carried out separately for each transformation rule.

Proposition 2 (Correctness) *Let (P_{nop}, P_{op}) be an EBG-admissible program pair, C be an EBG-admissible normal$^+$-clause $H \leftarrow B$, R be an SLDNF$^+$-*

refutation for $P_{in} \cup \{\leftarrow B\}$, where P_{in} is the normal program $P_{nop} \cup P_{op} \cup def(P_{op}) \cup \{eq(x,x)\leftarrow\}$. Suppose the clause $C^T = H^T \leftarrow B^T$ is the result of transformation $T(C, P_{nop}, P_{op}, R)$, where T is one of T_{nop}, $T_{\sim nop}$, $T_{\sim op}$, $T_{\sim\sim}$, or T_{eq}.

Suppose further that P'_{op} is a normal program such that (P_{nop}, P'_{op}) is an EBG-admissible program pair, and let P_{test} be the normal program $P_{nop} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x)\leftarrow\}$.

Then, for each SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow B^T\}$ with a computed answer σ^T, there is also an SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow B\}$ with a computed answer σ such that $H^T\sigma^T = H\sigma$.

Proof: Follows from lemmas 17 - 21 below. □

Lemma 17 *proposition 2 holds for $T = T_{nop}$.*

Proof: According to the definition of transformation T_{nop}, C is a clause $H \leftarrow L_1, \ldots, L_m$, there is a variant $Q = A \leftarrow A_1, \ldots, A_n$ of a program clause in P_{nop}, θ is an mgu for A and L_k (for some $k \in \{1, \ldots, m\}$), and C^T is the clause $(H \leftarrow L_1, \ldots, L_{k-1}, A_1, \ldots, A_n, L_{k+1}, \ldots, L_m)\theta$.

Then, if $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{k-1}, A_1, \ldots, A_n, L_{k+1}, \ldots, L_m)\theta\}$ has an SLDNF$^+$-refutation, we can add one step at the beginning to obtain and SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{k-1}, L_k, L_{k+1}, \ldots, L_m)\}$, in which L_k is selected; the first input clause is Q, and the first mgu is θ. Furthermore, the computed answer for H in the new refutation is the same as that for $H\theta$ in the old one. □

Lemma 18 *proposition 2 holds for $T = T_{\sim nop}$.*

Proof: According to the definition of transformation $T_{\sim nop}$, C is a clause $H \leftarrow M_1, \ldots, M_k, \ldots, M_m$, where M_k has the form $\sim (L_1, \ldots, L_s, \ldots, L_n)$; $\{H_i \leftarrow B_i | i \in \{1, \ldots, q\}\}$ is a set of variants of the program clauses in P_{nop} for the predicate symbol in L_s, which have no variables in common with C; for each $i \in \{1, \ldots, q\}$, N_i is $\sim (L_1, \ldots, L_{s-1}, eq(L_s, H_i), B_i, L_{s+1}, \ldots, L_n)$, and C^T is the clause $H \leftarrow B^T$ with $B^T = M_1, \ldots, M_{k-1}, N_1, \ldots, N_q, M_{k+1}, \ldots, M_m$.

Suppose there is an SLDNF$^+$-refutation S for $P_{test} \cup \{\leftarrow B^T\}$ with a sequence of goals $G_0 = \leftarrow B^T, \ldots, G_p = \Box$ and a sequence of mgu's $\theta_1, \ldots \theta_p$. It follows from the definition of an SLDNF$^+$-refutation that for each $i \in \{1, \ldots, q\}$, if G_{l_i} is the goal in this sequence in which N_i is selected, then no variable in $N_i\theta_1 \ldots \theta_{l_i}$ is bindable in G_{l_i}; hence also $N_i\theta_1 \ldots \theta_{l_i}\theta_{l_i+1} \ldots \theta_p = N_i\theta_1 \ldots \theta_{l_i}$. Note that $H_i\theta_1 \ldots \theta_p = H_i$, because $\mathrm{var}(H_i) \cap \mathrm{var}(C) = \emptyset$. It follows from lemma 3 that for all $i \in \{1, \ldots, q\}$, if $L_s\theta_1 \ldots \theta_p$ and H_i are unifiable with mgu σ_i, then $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, B_i, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_p\sigma_i\}$ has a finitely failed SLDNF-tree T_i. These are the subtrees of the root in a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_n)\theta_1 \ldots \theta_p\}$, since P'_{op} does not contain clauses for the predicate in L_s.

We obtain an SLDNF$^+$-refutation S' for $P_{test} \cup \{\leftarrow M_1, \ldots, M_m\}$ from S by leaving out steps in which (instances of) the literals $N_1, \ldots, N_q$ are selected, and selecting $M_k\theta_1 \ldots \theta_p$ in the last step. Obviously, the answer substitution for H computed by S' is the same as that one computed by S. $\Box$

Lemma 19 *proposition 2 holds for* $T = T_{\sim op}$.

Proof: According to the definition of transformation $T_{\sim op}$, C is a clause $H \leftarrow M_1, \ldots, M_k, \ldots, M_m$, where M_k has the form $\sim (L_1, \ldots, L_s, \ldots, L_n)$, and P_{nop} does not contain clauses for the predicate symbol p of L_s. Suppose that $C^T = H \leftarrow B^T$ is the transformed clause, and that $P_{test} \cup \{\leftarrow B^T\}$ has an SLDNF$^+$-refutation S with a sequence of goals $G_0 = \leftarrow B^T, \ldots, G_r = \Box$ and a sequence of mgu's $\theta_1, \ldots \theta_r$.

If cases a) or b) in the transformation apply, the claim follows immediately from lemma 1. Therefore now consider condition c). Then B^T has the form $M_1, \ldots, M_{k-1}, \mathit{definition}(p, [x_1, \ldots, x_q]), N_1, \ldots, N_q, M_{k+1}, \ldots, M_m$, where $x_1, \ldots, x_q$ are distinct variables not occurring in C, and for each $i \in \{1, \ldots, q\}$, N_i is the term $\sim (L_1, \ldots, L_{s-1}, eq(L_s, x_i), L_{s+1}, \ldots, L_n)$.

It follows from the definition of an SLDNF$^+$-refutation that for each $i \in \{1, \ldots, q\}$, if G_{l_i} is the goal in this sequence in which N_i is selected, then no variable in $N_i\theta_1 \ldots \theta_{l_i}$ is bindable in G_{l_i}; hence also $N_i\theta_1 \ldots \theta_{l_i} = N_i\theta_1 \ldots \theta_{l_i}\theta_{l_i+1} \ldots \theta_r$.

Suppose the literal $\mathit{definition}(p, [x_1, \ldots, x_q])$ is selected in G_l. Then, according to the construction of $\mathit{def}(P'_{op})$, P'_{op} contains exactly q clauses $H_1 \leftarrow$, $\ldots, H_q \leftarrow$ for p, and θ_{l+1} is a substitution $\{x_i/H_i | i \in \{1, \ldots, q\}\}$. Note that $N_i\theta_1 \ldots \theta_r = N_i\theta_1 \ldots \theta_l\{x_i/H_i\}\theta_{l+2} \ldots \theta_r$ (since each x_j does not occur in N_i, for $j \neq i$) $= L_1, \ldots, L_{s-1}, eq(L_s, H_i), L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\theta_{l+2} \ldots \theta_r$, since x_i does not occur in C.

It follows from lemma 3 that for all $i \in \{1, \ldots, q\}$, if $L_s\theta_1 \ldots \theta_l\theta_{l+2} \ldots \theta_r$ and H_i are unifiable with mgu σ_i, then there is a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l \theta_{l+2} \ldots \theta_r \sigma_i$. Since P_{nop} does not contain clauses for p, these are the subtrees of the root in a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_s, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l \theta_{l+2} \ldots \theta_r\}$.

We obtain an SLDNF$^+$-refutation S' for $P_{test} \cup \{\leftarrow M_1, \ldots, M_m\}$ from S by leaving out steps in which (instances of) the literals $\mathit{definition}(p, [x_1, \ldots, x_q])$, $N_1, \ldots, N_q$ are selected, and selecting $M_k\theta_1 \ldots \theta_l \theta_{l+2} \ldots \theta_r$ in the last step. Since $x_1, \ldots, x_q$ do not occur in H, the answer substitution for H computed by S' is the same as that one computed in S. $\Box$

Lemma 20 *proposition 2 holds for* $T = T_{\sim\sim}$.

Proof: According to the definition of transformation $T_{\sim\sim}$, C is a clause $H \leftarrow M_1, \ldots, M_k, \ldots, M_m$, where M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$, and L_s is $\sim A_s$. The transformed clause C^T has the form $H \leftarrow B^T$, where B^T is $M_1, \ldots, M_{k-1}, M'_k, M_{k+1}, \ldots, M_m$, and M'_k is either $\sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)$ (a) or A_s (b).

Suppose there is an SLDNF$^+$-refutation S for $P_{test} \cup \{\leftarrow B^T\}$ with a sequence of goals $G_0 = \leftarrow B^T, \ldots, G_p = \Box$ and a sequence of mgu's $\theta_1, \ldots \theta_p$.

In case a), M'_k is selected in some goal G_l, and there is a finitely failed SLDNF-tree T for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$. According to lemma 1, also $P_{test} \cup \{\leftarrow L_1, \ldots, L_{s-1}, L_s, L_{s+1}, \ldots, L_n)\theta_1 \ldots \theta_l\}$ has a finitely failed SLDNF-tree; hence, an SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow M_1, \ldots, M_m\}$ with the same sequence of mgu's can be constructed from S which selects $M_k\theta_1 \ldots \theta_l$ in step $l+1$ instead of $M'_k\theta_1 \ldots \theta_l$.

In case b), there must be an SLDNF$^+$-refutation S' for $P_{test} \cup \{\leftarrow B^T\theta_1 \ldots \theta_p\}$ with a sequence of mgu's $\theta'_1, \ldots, \theta'_p$ such that $C\theta_1 \ldots \theta_p \theta'_1 \ldots \theta'_p = C\theta_1 \ldots \theta_p$. By selecting a subsequence of S', an SLDNF$^+$-refutation S'' for $P_{test} \cup \{\leftarrow (M_1, \ldots, M_{k-1}, M_{k+1}, \ldots M_m)\theta_1 \ldots \theta_p\}$ can be constructed, with a sequence of mgu's $\theta''_1, \ldots, \theta''_r$ such that $C\theta_1 \ldots \theta_p \theta''_1 \ldots \theta''_r = C\theta_1 \ldots \theta_p$.

The variables in $\theta_1 \ldots \theta_p$ which occur in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$ are bindable in $M_1, \ldots, M_{k-1}, A_s, M_{k+1}, \ldots, M_m$. According to the definition of a finitely failed SLDNF-tree, the instance of L_s which was selected in the example refutation R must have been ground; hence, all variables in A_s must be bindable in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$. Consequently, all variables in $\theta_1 \ldots \theta_p$ that occur in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$ are also bindable in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$.

By corollary 3, $P_{test} \cup \{\leftarrow M_1, \ldots, M_{k-1}, M_{k+1}, \ldots M_m\}$ is allowed, and corollary 2 yields an SLDNF$^+$-refutation S''' with a sequence of mgu's $\theta'''_1, \ldots, \theta'''_r$, such that $C\theta_1''' \ldots \theta_r''' = C\theta_1 \ldots \theta_p\theta_1'' \ldots \theta_r'' = C\theta_1 \ldots \theta_p$. Particularly, $H\theta_1''' \ldots \theta_r''' = H\theta_1 \ldots \theta_p$, and $L_s\theta_1''' \ldots \theta_r''' = L_s\theta_1 \ldots \theta_p$ is ground; consequently, a finitely failed SLDNF-tree exists for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_s, \ldots, L_n)\theta_1''' \ldots \theta_r'''\}$. By adding one step at the end of S''' in which $M_k\theta_1'' \ldots \theta_r''$ is selected, we obtain an SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow M_1, \ldots, M_m\}$. $\Box$

Lemma 21 *proposition 2 holds for $T = T_{eq}$.*

Proof: Suppose C is a clause $H \leftarrow M_1, \ldots, M_k, \ldots, M_m$, where M_k has the form $\sim(L_1, \ldots, L_s, \ldots, L_n)$, and L_s has the form $eq(t_1, t_2)$. If case a) in the definition of transformation T_{eq} holds, the claim is trivial. Case b) follows immediately from lemma 1. Therefore assume now that case c) applies.

Then π is a renaming substitution for all variables in L_s which are not bindable in $M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$, σ is an mgu for $t_1\pi$ and $t_2\pi$, and $C^T = H^T \leftarrow B^T$ is $(H \leftarrow M_1, \ldots, M_{k-1}, M_k', M_{k+1}, \ldots, M_m)\sigma$, where M_k' is $\sim(L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\pi$.

Suppose $P_{test} \cup \{\leftarrow B^T\}$ has an SLDNF$^+$-refutation with a sequence of goals $G_0 = \leftarrow B^T, G_1, \ldots, G_p = \Box$ and a sequence of mgu's $\theta_1, \ldots \theta_p$. Then M_k' is selected in some goal G_l, and there is a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\pi\sigma\theta_1 \ldots \theta_l\}$. According to the definition of an SLDNF$^+$-refutation, no variable in $M_k'\sigma\theta_1 \ldots \theta_l$ is bindable in G_l, therefore $M_k'\sigma\theta_1 \ldots \theta_l = M_k'\sigma\theta_1 \ldots \theta_l\theta_{l+1} \ldots \theta_p$.

Let G' be the goal $\leftarrow M_1, \ldots, M_{k-1}, M_{k+1}, \ldots, M_m$. $P \cup \{G'\sigma\}$ is allowed and has an SLDNF$^+$-refutation with a sequence of mgu's $\theta_1, \ldots, \theta_l, \theta_{l+2}, \ldots, \theta_p$. Since all variables in $[t_1, t_2]\pi$ occurring in G' are bindable in G', lemma 8 yields an SLDNF$^+$-refutation S for $P \cup \{G'\}$ of the same length with a sequence of mgu's $\theta_1', \ldots, \theta_{p-1}'$ such that $H\sigma\theta_1 \ldots \theta_p = H\theta_1' \ldots \theta_{p-1}'$; moreover, an mgu μ exists for $t_1\pi\theta_1' \ldots \theta_{p-1}'$ and $t_2\pi\theta_1' \ldots \theta_{p-1}'$ such that $M_k'\theta_1' \ldots \theta_{p-1}'\mu$ is a variant of $M_k'\sigma\theta_1 \ldots \theta_p$. Thus, there is a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, L_{s+1}, \ldots, L_n)\pi\theta_1' \ldots \theta_{p-1}'\mu\}$.

According to lemma 3, there is also a finitely failed SLDNF-tree for $P_{test} \cup \{\leftarrow (L_1, \ldots, L_{s-1}, eq(t_1, t_2), L_{s+1}, \ldots, L_n)\pi\theta_1' \ldots \theta_{p-1}'\}$. Since $\mathrm{var}(\pi) \cap \mathrm{var}(\theta_1' \ldots \theta_{p-1}') = \emptyset$, this goal is a variant of $\leftarrow (L_1, \ldots, L_{s-1}, eq(t_1, t_2), L_{s+1}, \ldots, L_n)\theta_1' \ldots \theta_{p-1}'$; consequently, by adding one step at the end of S in which $M_k\theta_1' \ldots \theta_{p-1}'$ is selected, an SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow M_1, \ldots, M_m\}$ can be constructed. $\Box$

4.6 Summary

Now we are in a position to combine the results of the preceding sections to prove that the transformation system satisfies the requirements of EBG, as it was formalized in section 4.2. For convenience, this statement is repeated.

Theorem 1 *Let (P_{nop}, P_{op}) be an EBG-admissible program pair, and A be an atom. Suppose that $P_{nop} \cup P_{op} \cup \{\leftarrow A\}$ has an SLDNF-refutation with computed answer σ. Then there is a transformation sequence for $(A\sigma, P_{nop}, P_{op})$ with a resulting clause C such that the following conditions hold:*

a) *C is operational wrt P_{op}.*

b) *$\{C\} \cup P_{op} \cup def(P_{op}) \cup \{eq(x,x) \leftarrow\}$ is sufficient for A and σ.*

c) *For each normal program P'_{op} such that (P_{nop}, P'_{op}) is an EBG-admissible program pair, $\{C\} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x) \leftarrow\}$ is correct wrt $P_{nop} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x) \leftarrow\}$.*

Proof: Suppose A has the form $p(t_1, \ldots, t_n)$; let A' be the atom $p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are distinct variables not occurring in $P_{in} \cup \{\leftarrow A\}$. According to corollary 3 and corollary 2, there is an SLDNF-refutation R_1 for $P_{nop} \cup P_{op} \cup \{\leftarrow A'\}$ with computed answer σ_1 such that $A\sigma = A'\sigma_1$. Note that since the predicates $eq/2$ and $definition/2$ do not occur in $P_{nop} \cup P_{op}$, R_1 is also an SLDNF$^+$-refutation for $P_{in} \cup \{\leftarrow A'\}$, where P_{in} is the normal program $P_{nop} \cup P_{op} \cup \mathrm{def}(P_{op}) \cup \{eq(x,x) \leftarrow\}$. This is the first refutation in the transformation sequence, and the first clause C_1 is $A' \leftarrow A'$.

Proposition 1 shows that for each EBG-admissible normal$^+$-clause $C_i = H_i \leftarrow B_i$ in the transformation sequence which is not operational wrt P_{op}, the preconditions of at least one transformation are satisfied, for any refutation R_i for $P_{in} \cup \{\leftarrow B_i\}$. Furthermore, a clause $C_{i+1} = H_{i+1} \leftarrow B_{i+1}$ and a refutation R_{i+1} for $P_{in} \cup \{\leftarrow B_{i+1}\}$ can be constructed; if σ_i and σ_{i+1} are the answer substitutions computed by R_i and R_{i+1}, respectively, then $H_i\sigma_i = H_{i+1}\sigma_{i+1}$. According to lemma 11, C_{i+1} must also be EBG-admissible. Therefore, the transformation sequence does not stop with a non-operational clause.

Denote the number of literals in clause C_k as l_k. It follows from proposition 1 that if R_i has complexity $c(R_i)$, then R_{i+1} is of complexity $c(R_{i+1})$ such that $(c(R_i), l_i)$ is lexically greater than $(c(R_{i+1}), l_{i+1})$. But as $c(R_0)$ and all l_k

are finite, this well-founded ordering guarantees that the transformation sequence terminates with a clause $C_n = H_n \leftarrow B_n$ which is operational wrt P_{op}. (statement a). Moreover, by proposition 1, if σ_n is the answer substitution computed by R_n, then $H_n\sigma_n = A'\sigma_1$.

According to lemma 9, there is an SLDNF$^+$-refutation R^T for $\{C_n\} \cup P_{in} \cup \{\leftarrow A\}$ which uses C_n in the first step. If σ^T is the answer substitution computed by R^T, then $A\sigma^T = H_n\sigma_n = A'\sigma_1 = A\sigma$. Since C_n is operational wrt P_{op}, no predicate symbol defined in P_{nop} can occur in a literal of its body; consequently, R^T is also an SLDNF$^+$-refutation for $\{C_n\} \cup P_{op} \cup \text{def}(P_{op}) \cup \{eq(x,x) \leftarrow\} \cup \{\leftarrow A\}$ (statement b).

c) Now let P'_{op} be a normal program, such that (P_{nop}, P'_{op}) is an EBG-admissible program pair, and G be an atom. Suppose there is an SLDNF$^+$-refutation S for $\{C_n\} \cup P'_{op} \cup \text{def}(P'_{op}) \cup \{eq(x,x) \leftarrow\} \cup \{\leftarrow G\}$ with computed answer ρ. It suffices to confine attention to the case that exactly the first input clause in S is a variant of C_n. Since C_n is range-restricted, it follows from lemma 10 that there is an SLDNF$^+$-refutation S_2 for $P'_{op} \cup \text{def}(P'_{op}) \cup \{eq(x,x) \leftarrow\} \cup \{\leftarrow B_n\}$ with computed answer ρ_2 such that $H_n\rho_2 = G\rho$. Since C_n is operational wrt P_{op}, no predicate symbol defined in P_{nop} can occur in its body; consequently, S_2 is also an SLDNF$^+$-refutation for $P_{test} \cup \{\leftarrow B_n\}$, where P_{test} is the normal program $P_{nop} \cup P'_{op} \cup \text{def}(P'_{op}) \cup \{eq(x,x) \leftarrow\}$. By repeated application of proposition 2, there is an SLDNF$^+$-refutation S_3 for $P_{test} \cup \{\leftarrow A'\}$ with computed answer ρ_3, such that $A'\rho_3 = H_n\rho_2$. Note that S_3 is also an SLDNF$^+$-refutation for $P_{nop} \cup P'_{op} \cup \{\leftarrow A'\}$, since the predicate symbols $eq/2$ and $\mathit{definition}/2$ do not occur in $P_{nop} \cup P'_{op}$. By lemma 9, an SLDNF$^+$-refutation S_4 exists for $P_{nop} \cup P'_{op} \cup \{\leftarrow G\}$ with computed answer ρ_4, such that $G\rho_4 = A'\rho_3 = H_n\rho_2 = G\rho$. $\square$

The definition of correctness used so far has been purely procedural. However, we can use the correctness of SLDNF-refutation [Ll87] in the following way: Suppose P is a normal program consisting of allowed clauses, A is an atom, and R is an SLDNF$^+$-refutation for $P \cup \{\leftarrow A\}$ with a sequence of goals $G_0 = \leftarrow A, G_1, \ldots, G_n = \square$. Then for each negative literal L occurring in a goal G_i, all variables in L occur in a positive literal in G_i. It follows from lemma 6 that only ground negative literals are selected in R; thus, R is also an SLDNF-refutation. Since the predicates $eq/2$ and $\mathit{definition}/2$ do not occur in $P_{nop} \cup P'_{op}$, theorem 1 implies the following corollary.

Corollary 4 *Let P_{nop}, P_{op}, A, C be as in theorem 1. Then, for each normal program P'_{op} such that (P_{nop}, P'_{op}) is an EBG-admissible program pair, and each atom G of the same predicate symbol as A, if there is an SLDNF$^+$-refutation for $\{C\} \cup P'_{op} \cup def(P'_{op}) \cup \{eq(x,x) \leftarrow\} \cup \{\leftarrow G\}$ with computed answer σ, then $G\sigma$ is a logical consequence of* $\mathrm{comp}(P_{nop} \cup P'_{op})$.

5 Implementation

As mentioned in section 3.2, the efficiency of the derived clauses depends on the efficiency of the computation rule used in the example SLDNF$^+$-refutation. Therefore, it is essential that the transformation system allows for arbitrary computation rules. However, in order to make it comparable to other approaches, in this section we will restrict ourselves to PROLOG-derivations. Furthermore, in each step, among all possible transformations that one is chosen which applies to the leftmost literal.

Under these preconditions, the transformation system is implemented by the meta-interpreter listed below. There are close similarities to that one proposed in [BruHe88]. It is assumed that the bodies of program clauses are represented as lists; for the case of facts, the empty list [] is used. The operationality criterion has to be declared using the meta-predicate *operational*/1. Furthermore, the predicates *append*/3, *findall*/2 and *copy*/2 are supposed to work as expected [SteSha86].

```
(1)     solve(Goal,GenGoal,Leaves) ←
            solve(Goal,GenGoal,[],[],_,Leaves).

(2)     solve([],[],PosLeaves,Leaves,PosLeaves,Leaves).
(3)     solve(not(Goal),not(GenGoal),PosIn,LeavesIn,PosOut,LeavesOut) ←
            !,
            fail([Goal],[GenGoal],PosIn,LeavesIn,PosOut,LeavesOut).
(4)     solve([Goal|Goals],[GenGoal|GenGoals],PosIn,LeavesIn,PosOut,LeavesOut) ←
            !,
            solve(Goal,GenGoal,PosIn,LeavesIn,Pos1,Leaves1),
            solve(Goals,GenGoals,Pos1,Leaves1,PosOut,LeavesOut).
(5)     solve(Leaf,GenLeaf,PosIn,LeavesIn,PosOut,LeavesOut) ←
            operational(Leaf),
```

```
          !,
          solve(Leaf),
          append(PosIn,[GenLeaf],PosOut),
          append(LeavesIn,[GenLeaf],LeavesOut).
(6)   solve(Goal,GenGoal,PosIn,LeavesIn,PosOut,LeavesOut) ←
          clause(GenGoal,GenBody),
          copy((GenGoal←GenBody),(Goal←Body)),
          solve(Body,GenBody,PosIn,LeavesIn,PosOut,LeavesOut).


(7)   fail([not(Goal)|_],[not(GenGoal)|_],PosIn,LeavesIn,PosOut,LeavesOut) ←
          solve(Goal,GenGoal,PosIn,LeavesIn,PosOut,LeavesOut),
          !.
(8)   fail([not(_)|Goals],[not(_)|GenGoals],PosIn,LeavesIn,PosOut,LeavesOut) ←
          !,
          fail(Goals,GenGoals,PosIn,LeavesIn,PosOut,LeavesOut).
(9)   fail([Goal|_],[GenGoal|_],PosLeaves,LeavesIn,PosLeaves,LeavesOut) ←
          operational(Goal),
          \+ solve(Goal),
          !,
          append(LeavesIn,[not(GenGoal)],LeavesOut).
(10)  fail([Goal|Goals],[GenGoal|GenGoals],PosIn,LeavesIn,PosOut,LeavesOut) ←
          ground(Goal),
          solve(Goal),
          !,
          fail(Goals,GenGoals,PosIn,LeavesIn,PosOut,LeavesOut).
(11)  fail([Goal|Goals],GenGoal,PosIn,LeavesIn,PosOut,LeavesOut) ←
          functor(Goal,F,A),
          functor(EmptyGoal,F,A),
          all_clauses(EmptyGoal,Clauses,GenClauses,ClauseCond),
          append(PosIn,ClauseCond,Pos1),
          append(LeavesIn,ClauseCond,Leaves1),
          fail_with_clauses(Clauses,GenClauses,[Goal|Goals],GenGoal,
                  Pos1,Leaves1,PosOut,LeavesOut).

(12)  fail_with_clauses([],[],_,_,PosLeaves,Leaves,PosLeaves,Leaves).
```

```
(13)  fail_with_clauses([(H←B)|Clauses],[(HH←BB)|GenClauses],Goal,
                GenGoal,PosIn,LeavesIn,PosOut,LeavesOut) ←
          fail_with_clause((H←B),(HH←BB),Goal,GenGoal,
                PosIn,LeavesIn,Pos1,Leaves1),
          fail_with_clauses(Clauses,GenClauses,Goal,GenGoal,
                Pos1,Leaves1,PosOut,LeavesOut).

(14)  fail_with_clause((H←B),(HH←BB),_,[GenGoal|_],
                PosLeaves,Leaves,PosLeaves,Leaves) ←
          \+ (HH = GenGoal),
          !.
(15)  fail_with_clause((H←B),(HH←BB),[Goal|_],[GenGoal|_],
                PosLeaves,LeavesIn,PosLeaves,LeavesOut) ←
          \+ (H = Goal),
          !,
          append(LeavesIn, [not(GenGoal=HH)],LeavesOut).
(16)  fail_with_clause((H←B),(HH←BB),[Goal|Goals],[GenGoal|GenGoals],
            PosIn,LeavesIn,PosOut,LeavesOut) ←
          copy([Goal|Goals],[H|CopiedGoals]),
          append(B,CopiedGoals,DerivedGoal),
          copy(PosIn-[GenGoal|GenGoals], PosIn-[HH|CopiedGenGoals]),
          append(BB,CopiedGenGoals,DerivedGenGoal),
          fail(DerivedGoal,DerivedGenGoal,PosIn,LeavesIn,PosOut,LeavesOut).

(17)  all_clauses(Goal,Clauses,GenClauses,[findall(Goal,GenHeads)]) ←
          operational(Goal),
          !,
          findall((Goal←[]),solve(Goal),Clauses),
          copy_clauses(Clauses,Goal,GenClauses,GenHeads).
(18)  all_clauses(Goal,Clauses,GenClauses,[]) ←
          findall((Goal←Body), clause(Goal,Body),Clauses),
          copy(Clauses,GenClauses).

(19)  copy_clauses([],Goal,[],[]).
(20)  copy_clauses([C|Cs],Goal,[(HH←[])|GenClauses],[HH|Heads]) ←
          copy(Goal,HH),
```

copy_clauses(Cs,Goal,GenClauses,Heads).

Ignoring temporarily the third and fifth arguments of *solve*/6, clauses 2,4,5,6 are equivalent to the basic PROLOG-EBG-procedure of [KCMcC87]: *solve(Goal,GenGoal,PosLeavesIn,LeavesIn,PosLeavesOut,LeavesOut)* proves *Goal* and constructs in parallel a generalized proof tree for *GenGoal*; the list of its leaves is appended to *LeavesIn*, yielding *LeavesOut*. Clause 2 treats the case of the empty goal, in which the accumulated leaves are returned. Clause 4 evaluates conjunctive goals from left to right. For operational predicates, *GenGoal* is left unbound (clause 5), thereby abstracting from the instantiation of *Goal* which is specific to the example. The standard SLD-resolution step is handled by clause 6.

At each call, *LeavesIn* is a beginning part of the derived clause body, i.e. a list of all the literals which will occur left to the leaves of the proof tree for *GenGoal*. The other two additional arguments are necessary to keep track of the bindable variables in the derived clause, which are required for the renaming process corresponding to case c) of transformation T_{eq}. Hence, *PosLeavesIn* is always the subset of positive literals (i.e. atoms) in *LeavesIn*; correspondingly, *PosLeavesOut* returns the subset of atoms in *LeavesOut*.

When a negative literal is encountered, clause 3 gives rise to a call of *fail*/6, whose arguments have basically the same meaning as in *solve*/6: *fail(Goal,GenGoal,PosIn,LeavesIn,PosOut,LeavesOut)* succeeds if the conjunctive *Goal* finitely fails, and appends a list of literals sufficient to explain the failure of *GenGoal* to *LeavesIn*, yielding *LeavesOut*; otherwise it fails itself (note that this is especially true for the empty goal, designated as []).

Clauses 7 and 8 correspond to cases a) and b) of transformation $T_{\sim\sim}$, respectively. Clause 9 implements case a) of $T_{\sim op}$. Addition of a literal *operational*(*Goal*) to the body of clause 10 would yield the second case in the same transformation. However, as the justification for rule 29 in [BruHe88] shows, we may use this slight generalization. Clause 11 uniformly treats case c) of $T_{\sim op}$, as well as $T_{\sim nop}$. The difference between them is reflected by the two clauses for the predicate *all_clauses(Goal,Clauses,GenClauses,ClauseCond)*, which computes the list of program *Clauses* whose head unifies with *Goal*: if this is non-operational (clause 17), then *GenClauses* is just a copy of *Clauses*, and *ClauseCond* is the empty list; otherwise (clause 18), if n proofs exist for *Goal*, then *GenClauses* is a list $[H_1 \leftarrow, \ldots, H_n \leftarrow]$ and *ClauseCond* is a singleton list with the literal *findall*(*Goal*, $[H_1, \ldots, H_n]$), where each H_i is a copy of *Goal* (produced by *copy_clauses*/4). This literal corresponds to the special literal *definition*/2 introduced by $T_{\sim op}$.

Clauses 12 and 13 iterate *fail_with_clause*/8 through the list of program *Clauses* for the predicate of *Goal*. clauses 14-16 correspond to cases a)-c) in the definition of transformation T_{eq}, respectively. Clause 16 was the only reason to carry along *PosIn*, the preceding positive literals in the derived rule. This argument is used to prevent the call of *copy*/2 from renaming those variables which are bindable in the new clause. Then *fail*/6 is recursively called in depth-first-manner for *DerivedGoal*, which is a child of [Goal|Goals] in the respective SLDNF-tree. These remarks conclude the discussion of the meta-interpreter.

6 Related work

As already mentioned in the introduction, the starting point of the present work has been the *feature construction problem* in game playing domains. Concepts which classify positions according to their game-theoretic value can easily be formulated by means of negation. Therefore, in contrast to previous approaches to Explanation-Based Learning in this domain, our method is able to derive sufficient and correct (although possibly lengthy) classification rules for arbitrary games and positions.

In the realm of Partial Evaluation, correctness results for program transformation systems have already been established (cf. [KaKa88], [Ll91], [TaSa84]). These concern procedural (as in the theorem above), as well as declarative (i.e. model-based) semantics. However, the expansion of negative subgoals is only considered for limited cases, since its variables generally do not become fully instantiated at partial evaluation time [ChaWa89].

A number of meta-interpreters with (in some cases interactive) explanation facilities have been developed, which can also handle negations; One representative which builds complete proof trees without backtracking is the work of [BruHe88], which was already referred to in section 5. Yet their focus is not on *generalization*; thus, they do not distinguish between operational and non-operational predicates; there is no reduction of unifications (corresponding to transformation T_{eq}); the role of different computation rules is not considered.

7 Conclusion

We have reported an approach which extends Explanation-Based Generalization to domain theories containing negations in clause bodies. This extra expressiveness is necessary e.g. in game playing applications in order to characterize positions according to their game-theoretic value. Thus, our method generalizes previous approaches in this domain. The presentation in the form of a transformation system includes the possibility to conceive operationality criteria different from the one used here, as well as to integrate additional transformation rules. For the case of allowed domain theories and training examples, we have shown that the standard requirements of EBG are satisfied, namely operationality, sufficiency and correctness of the derived clause. Furthermore, a meta-interpreter implementation is given.

Acknowledgements

The author thanks Prof. Thomas Ottmann and Prof. Michael M. Richter for helpful discussions and suggestions which improved the readability of the report. He is supported by a grant of the Deutsche Forschungsgemeinschaft.

References

[BruHe88] Bruffaerts, A., and Henin, E., *Proof Trees for Negation as Failure: Yet Another Prolog Meta-Interpreter*, in: Kowalski, R.A., and Bowen, K.A. (eds), Proc. of the 5th International Conference and Symposium on Logic Programming, The MIT Press, (1988), pp 343-358

[BruSch89] Bruynooghe, M., and De Schreye, D., *Some Thoughts on the Role of Examples in Program Transformation and its Relevance for Explanation-Based Learning*, in: Jantke, K.P. (ed), Proc. of the International Workshop on Analogical and Inductive Inference, Springer Verlag Berlin etc., (1989), pp 60-77

[BruRa89] Bruynooghe, M., De Raedt, L., and De Schreye, D., *Explanation-Based Program Transformation*, Proc. IJCAI-89, Detroit MI, (1989), pp 407-412

[ChaWa89] Chan, D., and Wallace, M., *A Treatment of Negation during Partial Evaluation*, in: Abramson, H., and Rogers, M.H. (eds), Meta-

Programming in Logic Programming, The MIT Press, Cambridge MA, (1989), pp 299-317

[Cla78] Clark, K.L., *Negation as Failure*, in: Gallaire, H., and Minker, J. (eds), Logic and Data Bases, Plenum Press, NY, (1978), 293-322

[FlaDi86] Flann, N.S., and Dietterich, T.G., *Selecting Appropriate Representations for Learning from Examples*, Proc. AAAI-86, (1986), pp 460-466

[Gup87] Gupta, A., *Explanation-Based Failure Recovery*, Proc. AAAI-87, (1987), pp 606-610

[HaBu88] Harmelen, F.v., and Bundy, A., *Explanation-Based Generalization = Partial Evaluation*, Artificial Intelligence 36(1988):401-412

[Hi87] Hirsh, H., *Explanation-based Generalization in a Logic Programming Environment*, Proc. IJCAI-87, Milan, (1987), pp 221-227

[KaKa88] Kawamura, T., and Kanamori, T., *Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation*, ICOT Technical Report, Tokyo, (1988)

[KCMcC87] Kedar-Cabelli, S.T., and McCarty, L.T., *Explanation-Based Generalization as Theorem Proving*, Proc. of the Fourth International Workshop on Machine Learning, Irvine, (1987)

[Ll87] Lloyd, J.W., *Foundations of Logic Programming*, Second, Extended Edition, Springer-Verlag, (1987)

[Ll91] Lloyd, J.W., and Shepherdson, J.C., *Partial Evaluation in Logic Programming*, Journal of Logic Programming 11, Elsevier, NY, (1991), pp 217-242

[Min84] Minton, S., *Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples*, Proc. AAAI-84, (1984), 251-254

[MiCa87] Minton, S., and Carbonell, J.G., *Strategies for Learning Search Control Rules: An Explanation-Based Approach*, Proc. IJCAI-87, Milan, (1987), pp 228-235

[MKKC86] Mitchell, T., Keller, R., and Kedar-Cabelli, S., *Explanation-Based Generalization: A Unifying View*, Machine Learning 1:1, (1986), pp 47-80

[MoBha87] Mostow, J., and Bhatnagar, N., *Failsafe: A Floor Planner that Uses EBG to Learn from its Failures*, Proc. IJCAI-87, Milan, (1987), pp. 249-255

[SiPu88] Siqueira, J.L., and Puget, J.F., *Explanation-Based Generalization of Failures*, Proc. ECAI-88, (1988), pp 339-344

[SteSha86] Sterling, L., and Shapiro, E., *The Art of Prolog: Advanced Programming Techniques*, The MIT Press, Cambridge MA, (1986)

[Ta89] Tadepalli, P., *Lazy Explanation-Based Learning: A Solution to the Intractable Theory Problem*, Proc. IJCAI-89, Detroit MI, (1989), 694-700

[TaSa84] Tamaki, H., and Sato, T., *Unfold/Fold Transformations of Logic Programs*, Proc. of the Second International Conference on Logic Programming, Uppsala, (1984), pp 127-138

[Ur91] Urbano, P., *Learning by Explanation of Failures*, in: Kodratoff, Y. (ed), Proc. of the European Working Session on Learning, Springer Verlag, Berlin, (1991), pp 327-343

[YSUB90] Yee, R.C., Saxena, S., Utgoff, P.E., Barto, A.G., *Explaining Temporal Differences to Create Useful Concepts for Evaluating States*, Proc. AAAI-90, (1990), pp 882-888