Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2012-12

RWTH Aachen · Department of Computer Science · September 2012 (revised)

Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs[*]

Jürgen Giesl
LuFG Informatik 2, RWTH Aachen University, Germany

Thomas Ströder
LuFG Informatik 2, RWTH Aachen University, Germany

Peter Schneider-Kamp
Dept. of Mathematics and Computer Science, University of Southern Denmark

Fabian Emmes
LuFG Informatik 2, RWTH Aachen University, Germany

Carsten Fuhs
Dept. of Computer Science, University College London, UK

ABSTRACT

There exist many powerful techniques to analyze *termination* and *complexity* of *term rewrite systems* (TRSs). Our goal is to use these techniques for the analysis of other programming languages as well. For instance, approaches to prove termination of definite logic programs by a transformation to TRSs have been studied for decades. However, a challenge is to handle languages with more complex evaluation strategies (such as Prolog, where predicates like the *cut* influence the control flow). In this paper, we present a general methodology for the analysis of such programs. Here, the logic program is first transformed into a *symbolic evaluation graph* which represents all possible evaluations in a finite way. Afterwards, different analyses can be performed on these graphs. In particular, one can generate TRSs from such graphs and apply existing tools for termination or complexity analysis of TRSs to infer information on the termination or complexity of the original logic program.

Categories and Subject Descriptors

D.1.6 [**Programming Techniques**]: Logic Programming; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Mechanical Verification*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Automatic Analysis of Algorithms*

General Terms

Languages, Theory, Verification

Keywords

logic programs, rewriting, termination, complexity, determinacy

1. INTRODUCTION

We are concerned with analyzing "semantical" properties of logic programs, like *termination*, *complexity*, and *determinacy* (i.e., the question whether all queries in a specific class succeed at most once). While there are techniques and tools that analyze logic programs *directly*, we present a general *transformational* methodology for such analyses. In this way, one can re-use existing powerful techniques and tools that have been developed for *term rewriting*.

For well-moded definite logic programs, there are several transformations to TRSs such that termination of the TRS implies termination of the original logic program [32]. We extended these transformations to arbitrary definite programs in [34].

However, Prolog programs typically use the *cut* predicate. To handle the non-trivial control flow induced by cuts, in [36] we introduced a pre-processing method where a Prolog program is first transformed into a *symbolic evaluation graph*. (These graphs were inspired by related approaches to program optimization [37] and were called "termination graphs" in [36].) Symbolic evaluation graphs also represent those aspects of the program that cannot easily be expressed in term rewriting. We also developed similar approaches for other programming languages like Java and Haskell [7, 8, 9, 17]. For Prolog, the transformation from the program to the symbolic evaluation graph relies on a new "linear" operational semantics which we presented in [40]. From the symbolic evaluation graph, one can then generate a simpler program (without cuts) whose termination implies termination of the original Prolog program. In [36] we generated definite logic programs from the graph (whose termination could then be analyzed by transforming them further to TRSs, for example). In [39], we presented a more powerful approach which generates so-called *dependency triples* [30, 35] from the graph. These dependency triples are an adaption of the *dependency pair* technique [16], which is one of the most powerful methods for automated termination analysis of TRSs.

In the current paper, we show that the symbolic evaluation graph cannot only be used for termination analysis, but it is also very suitable as the basis for several other analyses, such as complexity or determinacy analysis. So symbolic evaluation graphs and term rewriting can be seen as a general methodology for the analysis of programming languages like Prolog.[1]

After recapitulating the underlying operational semantics in Sect. 2, we introduce the symbolic evaluation graph in Sect. 3. To use this graph for different forms of program

[*]Supported by the DFG under grant GI 274/5-3, the DFG Research Training Group 1298 (*AlgoSyn*), and the Danish Council for Independent Research, Natural Sciences.

[1]This methodology can also be used to analyze programs in other languages. For example, in [8] we used similar graphs not just for termination proofs, but also for disproving termination and for detecting `NullPointerException`s in Java programs.

analysis, we present several new theorems which express the connection between the "abstract evaluations" represented in the graph and the "concrete evaluations" of actual queries.

In Sect. 4, we present a new improved approach for termination analysis of logic programs, where one directly generates *term rewrite systems* from the symbolic evaluation graph. This results in a substantially more powerful approach than [36]. Compared to [39], our new approach is considerably simpler and it allows us to apply *any* tool for termination of TRSs when analyzing the termination of logic programs. So one does not need tools that handle the (non-standard) notion of "dependency triples" anymore.

In Sect. 5 we show that symbolic evaluation graphs and the TRSs generated from the graphs can also be used in order to analyze the *complexity* of logic programs. Here, we rely on recent results which show how to adapt techniques for termination analysis of TRSs in order to prove asymptotic upper bounds for the runtime complexity of TRSs automatically.

Finally, Sect. 6 demonstrates that the symbolic evaluation graph can also be used to analyze whether a class of queries is *deterministic*. Besides being interesting on its own, such a determinacy analysis is also needed in our new approach for complexity analysis of logic programs in Sect. 5.

We implemented all our contributions in our automated termination tool AProVE [15] and performed extensive experiments to compare our approaches with existing analysis techniques which work directly on logic programs. It turned out that our approaches for termination and complexity clearly outperform related existing techniques. For determinacy analysis, our approach can handle many examples where existing methods fail, but there are also many examples where the existing techniques are superior. Thus, here it would be promising to couple our approach with existing ones. All proofs can be found in the appendix.

2. PRELIMINARIES AND OPERATIONAL SEMANTICS OF PROLOG

See, e.g., [2] for the basics of logic programming. We label individual cuts to make their scope explicit. Thus, we use a signature Σ containing $\{!_m/0 \mid m \in \mathbb{N}\}$ and all predicate and function symbols. As in the ISO standard for Prolog [22], we do not distinguish between predicate and function symbols and just consider *terms* $\mathcal{T}(\Sigma, \mathcal{V})$ and no atoms. A *query* is a sequence of terms. Let $Query(\Sigma, \mathcal{V})$ denote the set of all queries, where $\square$ is the empty query. A *clause* is a pair $h \;{:}{-}\; B$ where the *head* h is a term and the *body* B is a query. If B is empty, then we write just "h" instead of "$h \;{:}{-}\; \square$". A *logic program* $\mathcal{P}$ is a finite sequence of clauses.

We now briefly recapitulate our operational semantics from [40], which is equivalent to the ISO semantics in [22]. As shown in [40], both semantics yield the same answer substitutions, the same termination behavior, and the same complexity. The advantage of our semantics is that it is particularly suitable for an extension to *classes* of queries, i.e., for the symbolic evaluation of *abstract* states, cf. Sect. 3. This makes our semantics particularly well suited for analyzing logic programs.

Our semantics is given by a set of inference rules that operate on *states*. A *state* has the form $(G_1 \mid \ldots \mid G_n)$ where each G_i is a *goal*. Here, G_1 represents the current query and $(G_2 \mid \ldots \mid G_n)$ represents the queries that have to be considered next. This backtrack information is contained in the state in order to describe the effect of cuts. Since each state contains all backtracking goals, our semantics is *linear* (i.e., an *evaluation* with these rules is just a sequence of states and not a search tree as in the ISO semantics).

Essentially, a *goal* is just a query, i.e., a sequence of terms. But to compute answer substitutions, a goal is labeled by a substitution which collects the unifiers used up to now. So if $(t_1, \ldots, t_k)$ is a query, then a goal has the form $(t_1, \ldots, t_k)_\theta$ for a substitution θ. In addition, a goal can also be labeled by a clause c, where $(t_1, \ldots, t_k)^c_\theta$ means that the next resolution has to be performed with clause c. Moreover, a goal can also be a *scope marker* $?_m$ for $m \in \mathbb{N}$. This marker denotes the end of the scope of cuts $!_m$ labeled with m. Whenever a cut $!_m$ is reached, all goals preceding $?_m$ are discarded.

Def. 1 shows the inference rules for the part of Prolog defining definite logic programming and the cut. See [40] for the inference rules for full Prolog. Here, S and S' are states and the query Q may also be $\square$ (then "(t, Q)" is t).

Definition 1 (Operational Semantics).

$$\frac{\square_\theta \mid S}{S}\;(\textsc{Suc}) \qquad \frac{(t,Q)^{h\,{:}{-}\,B}_\theta \mid S}{(B\sigma, Q\sigma)_{\theta\sigma} \mid S}\;(\textsc{Eval})\; \textit{if } mgu(t,h) = \sigma$$

$$\frac{?_m \mid S}{S}\;(\textsc{Fail}) \qquad \frac{(t,Q)^{h\,{:}{-}\,B}_\theta \mid S}{S}\;(\textsc{Backtrack})\; \textit{if } t \not\sim h$$

$$\frac{(t,Q)_\theta \mid S}{(t,Q)^{c_1[!/!_m]}_\theta \mid \ldots \mid (t,Q)^{c_a[!/!_m]}_\theta \mid ?_m \mid S}\;(\textsc{Case})$$

where t is no cut or variable, m is fresh, and $Slice_{\mathcal{P}}(t) = (c_1, \ldots, c_a)$

$$\frac{(!_m, Q)_\theta \mid S \mid ?_m \mid S'}{Q_\theta \mid ?_m \mid S'}\;(\textsc{Cut})\; \textit{where } S \textit{ contains no } ?_m \qquad \frac{(!_m, Q)_\theta \mid S}{Q_\theta}\;(\textsc{Cut})\; \textit{where } S \textit{ contains no } ?_m$$

The Suc rule is applicable if the first goal of our sequence could be proved. Then we backtrack to the next goal in the sequence. Fail means that for the current m-th case analysis, there are no further backtracking possibilities. But the whole evaluation does not have to fail, since the state S may still contain further alternative goals which have to be examined.

To make the backtracking possibilities explicit, the resolution of a program clause with the first atom t of the current goal is split into two operations. The Case rule determines which clauses could be applied to t by slicing the program according to t's root symbol. Here, $Slice_{\mathcal{P}}(\mathsf{p}(t_1, \ldots, t_n))$ is the sequence of all program clauses "$h \;{:}{-}\; B$" from $\mathcal{P}$ where $root(h) = \mathsf{p}/n$. The variables in program clauses are renamed when this is necessary to ensure variable-disjointness with the states. Thus, Case replaces the current goal $(t, Q)_\theta$ by a goal labeled with the first such clause and adds copies of $(t, Q)_\theta$ labeled by the other potentially applicable clauses as backtracking possibilities. Here, the top-down clause selection rule is taken into account. The cuts in these clauses are labeled by a fresh mark $m \in \mathbb{N}$ (i.e., $c[!/!_m]$ is the clause c where all cuts ! are replaced by $!_m$), and $?_m$ is added at the end of the new backtracking goals to denote their scope.

Example 2. *Consider the following logic program.*

$$\begin{array}{rll}
\mathsf{star}(XS, [\,]) & \text{:-}\ !. & (1)\\
\mathsf{star}([\,], ZS) & \text{:-}\ !, \mathsf{eq}(ZS, [\,]). & (2)\\
\mathsf{star}(XS, ZS) & \text{:-}\ \mathsf{app}(XS, YS, ZS), \mathsf{star}(XS, YS). & (3)\\
\mathsf{app}([\,], YS, YS). & & (4)\\
\mathsf{app}([X \mid XS], YS, [X \mid ZS]) & \text{:-}\ \mathsf{app}(XS, YS, ZS). & (5)\\
\mathsf{eq}(X, X). & & (6)
\end{array}$$

Here, $\mathsf{star}(t_1, t_2)$ *holds iff* t_2 *results from repeated concatenation of* t_1*. So we have* $\mathsf{star}([1,2],[\,])$*,* $\mathsf{star}([1,2],[1,2])$*,* $\mathsf{star}([1,2],[1,2,1,2])$*, etc. The cut in rule (2) is needed for termination of queries of the form* $\mathsf{star}([\,],t)$*. For the query* $\mathsf{star}([1,2],[\,])$*, we obtain the following evaluation, where we omitted the labeling by substitutions for readability.*

$$\begin{array}{rl}
\mathsf{star}([1,2],[\,]) & \vdash_{\textsc{Case}}\\
\mathsf{star}([1,2],[\,])^{(1')} \mid \mathsf{star}([1,2],[\,])^{(2')} \mid \mathsf{star}([1,2],[\,])^{(3)} \mid ?_1 & \vdash_{\textsc{Eval}}\\
!_1 \mid \mathsf{star}([1,2],[\,])^{(2')} \mid \mathsf{star}([1,2],[\,])^{(3)} \mid ?_1 & \vdash_{\textsc{Cut}}\\
\square \mid ?_1 & \vdash_{\textsc{Suc}}\\
?_1 & \vdash_{\textsc{Fail}}\ \varepsilon
\end{array}$$

So the Case *rule results in a state which represents a case analysis where we first try to apply the* star*-clause (1). The state also contains the next backtracking goals, since when backtracking later on, we would use clauses (2) and (3). Here,* $(1')$ *denotes* $(1)[!/!_1]$ *and* $(2')$ *denotes* $(2)[!/!_1]$*.*

For a goal $(t,Q)_\theta^{h\,\text{:-}\,B}$, if t unifies[2] with the head h of the program clause, we apply Eval, which replaces t by the body B of the clause and applies the mgu σ to the result. Moreover, σ contributes to the answer substitution, i.e., we replace the label θ by $\theta\sigma$.

If t does not unify with h (denoted "$t \not\sim h$"), we apply the Backtrack rule. Then, h :- B cannot be used and we backtrack to the next goal in our backtracking sequence.

Finally, there are two Cut rules. The first rule removes all backtracking information on the level m where the cut was introduced. Since its scope is explicitly represented by $!_m$ and $?_m$, we have turned the cut into a *local* operation depending only on the current state. Note that $?_m$ must not be deleted as the current goal Q_θ could still lead to another cut $!_m$. The second Cut rule is used if $?_m$ is missing (e.g., if a cut $!_m$ is already in the initial query). We treat such states as if $?_m$ were added at the end of the state.

For each query Q, its corresponding *initial state* consists of just $(Q[!/!_1])_{id}$ (i.e., all cuts in Q are labeled by a fresh number like 1 and the goal is labeled by the identity substitution id). The query Q is *terminating* if all evaluations starting in its corresponding initial state are finite. Our inference rules can also be used to define *answer substitutions.*

Definition 3 (Answer Substitution). *Let* S *be a state with a single goal* Q_σ *(which may additionally be labeled by a clause* c*). We say that* θ *is an* answer substitution *for* S *if there is an evaluation from* S *to a state* $(\square_{\sigma\theta} \mid S_{\mathit{suffix}})$ *for a (possibly empty) state* S_{suffix} *(i.e.,* $(\square_{\sigma\theta} \mid S_{\mathit{suffix}})$ *is obtained by repeatedly applying rules from Def. 1 to* S*). Similarly,* θ *is an* answer substitution *for a query if it is an answer substitution for the query's initial state.*

[2]In this paper, we consider unification with occurs check. Our method could be extended to unification without occurs check, but we left this as future work since most programs do not rely on the absence or presence of the occurs check.

3. FROM PROLOG TO SYMBOLIC EVALUATION GRAPHS

We now explain the construction of *symbolic evaluation graphs* which represent all evaluations of a logic program for a certain *class* of queries. While we already presented such graphs in [36], here we introduce a new formulation of the corresponding abstract inference rules which is suitable for generating TRSs afterwards. Moreover, we present new theorems (Thm. 5, 8, and 10) which express the exact connection between abstract and concrete evaluations. These theorems will be used to prove the soundness of our analyses later on.

We consider classes of atomic queries described by a $\mathsf{p}/n \in \Sigma$ and a *moding function* $m : \Sigma \times \mathbb{N} \to \{in, out\}$. So m determines which arguments of a symbol are "inputs". The corresponding class of queries is $\mathcal{Q}_m^{\mathsf{p}} = \{\mathsf{p}(t_1, \ldots, t_n) \mid \mathcal{V}(t_i) = \varnothing$ for all i with $m(\mathsf{p}, i) = in\}$. Here, "$\mathcal{V}(t_i)$" denotes the set of all variables occurring in t_i. So for the program of Ex. 2, we might regard the class of queries $\mathcal{Q}_m^{\mathsf{star}}$ where $m(\mathsf{star}, 1) = m(\mathsf{star}, 2) = in$. Thus, $\mathcal{Q}_m^{\mathsf{star}} = \{\mathsf{star}(t_1, t_2) \mid t_1, t_2$ are ground$\}$.

To represent classes of queries, we regard *abstract* states that stand for sets of concrete states. Instead of "ordinary" variables $\mathcal{N}$, abstract states use abstract variables $\mathcal{A} = \{T_1, T_2, \ldots\}$ representing fixed, but arbitrary terms (i.e., $\mathcal{V} = \mathcal{N} \uplus \mathcal{A}$).

To obtain concrete states from an abstract one, we use *concretizations*. A concretization is a substitution γ which replaces all abstract variables by concrete terms, i.e., $\mathit{Dom}(\gamma) = \mathcal{A}$ and $\mathcal{V}(\mathit{Range}(\gamma)) \subseteq \mathcal{N}$. To determine by which terms an abstract variable may be instantiated, we add a knowledge base $KB = (\mathcal{G}, \mathcal{U})$ to each state, where $\mathcal{G} \subseteq \mathcal{A}$ and $\mathcal{U} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathcal{T}(\Sigma, \mathcal{V})$. The variables in $\mathcal{G}$ may only be instantiated by ground terms, i.e., $\mathcal{V}(\mathit{Range}(\gamma|_\mathcal{G})) = \varnothing$. Here, "$\gamma|_\mathcal{G}$" denotes the restriction of γ to $\mathcal{G}$, i.e., $\gamma|_\mathcal{G}(X) = \gamma(X)$ for $X \in \mathcal{G}$ and $\gamma|_\mathcal{G}(X) = X$ for $X \in \mathcal{V} \setminus \mathcal{G}$. A pair $(t, t') \in \mathcal{U}$ means that we are restricted to concretizations γ where $t\gamma \not\sim t'\gamma$, i.e., t and t' must not be unifiable after γ is applied. Then we say that γ is a concretization *w.r.t. KB*.

Thus, an abstract state has the form $(S; KB)$. Here, S has the form $(G_1 \mid \ldots \mid G_n)$ where the G_i are goals over the signature Σ and the abstract variables $\mathcal{A}$ (i.e., they do not contain variables from $\mathcal{N}$). In contrast to [36], we again label all goals (except scope markers) by substitutions $\theta : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{A})$ in order to store which substitutions were applied during an evaluation. These substitution labels will be necessary for the synthesis of TRSs in Sect. 4.

The notion of *concretization* can also be used for states. A (concrete) state S' is a concretization of $(S; KB)$ if there exists a concretization γ w.r.t. KB such that S' results from $S\gamma$ by replacing the substitution labels of its goals by arbitrary (possibly different) substitutions $\theta : \mathcal{N} \to \mathcal{T}(\Sigma, \mathcal{N})$. To ease readability, we often write "$S\gamma$" to denote an arbitrary concretization of $(S; KB)$. Let $\mathcal{CON}(S; KB)$ denote the set of all concretizations of an abstract state $(S; KB)$.

For a class $\mathcal{Q}_m^{\mathsf{p}}$ with p/n, now the *initial state* is $(\mathsf{p}(T_1, \ldots, T_n)_{id}, (\mathcal{G}, \varnothing))$, where $\mathcal{G}$ contains all T_i with $m(\mathsf{p}, i) = in$.

We now adapt the inference rules of Def. 1 to abstract states. The rules Suc, Fail, Cut, and Case do not change the knowledge base and are straightforward to adapt. In Def. 1, we determined which of the rules Eval and Backtrack to apply by trying to unify the first term t with the head h of the corresponding clause. But in the abstract case we might need to apply EVAL for some concretizations and BACKTRACK for others. The abstract BACKTRACK rule in Def. 4 can be used if $t\gamma$ does not unify with h for *any* concretization γ. Otherwise, $t\gamma$ unifies with h for *some* concretizations γ, but possibly not for *others*. Thus, the abstract EVAL rule has two successor states to combine both the concrete EVAL and the concrete BACKTRACK rule. Consequently, we now obtain symbolic evaluation trees instead of sequences.

DEFINITION 4 (ABSTRACT INFERENCE RULES).

$$\frac{(\square_\theta \,|\, S); KB}{S; KB}\ (\textsc{Suc}) \qquad \frac{((!_m, Q)_\theta \,|\, S \,|\, ?_m \,|\, S'); KB}{(Q_\theta \mid ?_m \mid S'); KB}\ (\textsc{Cut}) \ \textit{where } S \textit{ contains no } ?_m$$

$$\frac{(?_m \mid S); KB}{S; KB}\ (\textsc{Fail}) \qquad \frac{((!_m, Q)_\theta \mid S); KB}{Q_\theta; KB}\ (\textsc{Cut}) \ \textit{where } S \textit{ contains no } ?_m$$

$$\frac{((t, Q)_\theta \mid S); KB}{((t,Q)_\theta^{c_1[!/!_m]} \mid \ldots \mid (t,Q)_\theta^{c_a[!/!_m]} \mid ?_m \mid S); KB}\ (\textsc{Case})$$

where t is no cut or variable, m is fresh, $Slice_{\mathcal{P}}(t) = (c_1, \ldots, c_a)$

$$\frac{((t,Q)_\theta^{h\,:\text{-}\,B} \mid S); KB}{S; KB}\ (\textsc{Backtrack}) \ \textit{if there is no concretization } \gamma \textit{ w.r.t. } KB \textit{ such that } t\gamma \sim h.$$

$$\frac{((t,Q)_\theta^{h\,:\text{-}\,B} \mid S); (\mathcal{G},\mathcal{U})}{((B\sigma, Q\sigma)_{\theta\sigma} \mid S'); (\mathcal{G}', \mathcal{U}\sigma|_{\mathcal{G}}) \qquad S; (\mathcal{G}, \mathcal{U} \cup \{(t,h)\})}\ (\textsc{Eval})$$

if $mgu(t,h) = \sigma$. W.l.o.g., $\mathcal{V}(Range(\sigma))$ only contains fresh abstract variables and $Dom(\sigma)$ contains all previously occurring variables. Moreover, $\mathcal{G}' = \mathcal{A}(Range(\sigma|_{\mathcal{G}}))$ and S' results from S by applying the substitution $\sigma|_{\mathcal{G}}$ to its goals and by composing $\sigma|_{\mathcal{G}}$ with the substitution labels of its goals.

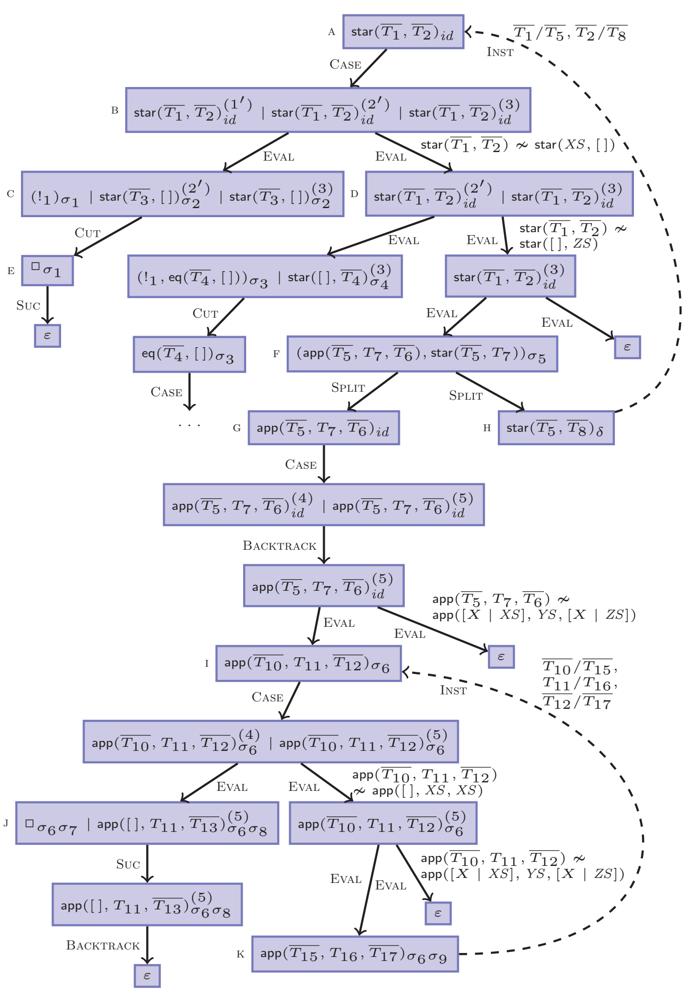
To handle "sharing" effects correctly [36], w.l.o.g. we assume that $mgu(t,h) = \sigma$ renames all occurring variables to fresh abstract variables in EVAL. The knowledge base is updated differently for the successors corresponding to the concrete EVAL and BACKTRACK rule. For all concretizations corresponding to the second successor of EVAL, the concretization of t does not unify with h. Hence, here we add (t,h) to $\mathcal{U}$.

Now consider concretizations γ where $t\gamma$ and h unify, i.e., these concretizations γ correspond to the first successor of the EVAL rule. Then for any $T \in \mathcal{G}$, $T\gamma$ is a ground instance of $T\sigma$. Hence, we replace all $T \in \mathcal{G}$ by $T\sigma$, i.e., we apply $\sigma|_{\mathcal{G}}$ to S. The new set $\mathcal{G}'$ of variables that may only be instantiated by ground terms are the abstract variables occurring in $Range(\sigma|_{\mathcal{G}})$ (denoted "$\mathcal{A}(Range(\sigma|_{\mathcal{G}}))$"). As before, t is replaced by the instantiated clause body B and the previous substitution label θ is composed with the mgu σ (yielding $\theta\sigma$).

Thm. 5 states that any concrete evaluation with Def. 1 can also be simulated with the abstract rules of Def. 4.

THEOREM 5 (SOUNDNESS OF ABSTRACT RULES). *Let $(S; KB)$ be an abstract state with a concretization $S\gamma \in \mathcal{CON}(S; KB)$, and let S_{next} be the successor of $S\gamma$ according to the operational semantics in Def. 1. Then the abstract state $(S; KB)$ has a successor $(S'; KB')$ according to an inference rule from Def. 4 such that $S_{next} \in \mathcal{CON}(S'; KB')$.*

As an example, consider the program from Ex. 2 and the class of queries $\mathcal{Q}_m^{\mathsf{star}}$. The corresponding initial state is



A $\mathsf{star}(\overline{T_1}, \overline{T_2})_{id}$
$\overline{T_1}/\overline{T_5}, \overline{T_2}/\overline{T_8}$
INST
CASE
B $\mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(1')} \mid \mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(2')} \mid \mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(3)}$
EVAL
EVAL
$\mathsf{star}(\overline{T_1}, \overline{T_2}) \nsim \mathsf{star}(XS, [\,])$
C $(!_1)_{\sigma_1} \mid \mathsf{star}(\overline{T_3}, [\,])_{\sigma_2}^{(2')} \mid \mathsf{star}(\overline{T_3}, [\,])_{\sigma_2}^{(3)}$
D $\mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(2')} \mid \mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(3)}$
CUT
EVAL
EVAL
$\mathsf{star}(\overline{T_1}, \overline{T_2}) \nsim \mathsf{star}([\,], ZS)$
E $\square_{\sigma_1}$
$(!_1, \mathsf{eq}(\overline{T_4}, [\,]))_{\sigma_3} \mid \mathsf{star}([\,], \overline{T_4})_{\sigma_4}^{(3)}$
$\mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(3)}$
SUC
CUT
EVAL
EVAL
ε
$\mathsf{eq}(\overline{T_4}, [\,])_{\sigma_3}$
F $(\mathsf{app}(\overline{T_5}, T_7, \overline{T_6}), \mathsf{star}(\overline{T_5}, T_7))_{\sigma_5}$
ε
CASE
SPLIT
SPLIT
...
G $\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})_{id}$
H $\mathsf{star}(\overline{T_5}, \overline{T_8})_\delta$
CASE
$\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})_{id}^{(4)} \mid \mathsf{app}(\overline{T_5}, T_7, \overline{T_6})_{id}^{(5)}$
BACKTRACK
$\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})_{id}^{(5)}$
$\mathsf{app}(\overline{T_5}, T_7, \overline{T_6}) \nsim \mathsf{app}([X \mid XS], YS, [X \mid ZS])$
EVAL
EVAL
ε
I $\mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}})_{\sigma_6}$
$\overline{T_{10}}/\overline{T_{15}}, T_{11}/T_{16}, \overline{T_{12}}/\overline{T_{17}}$
INST
CASE
$\mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}})_{\sigma_6}^{(4)} \mid \mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}})_{\sigma_6}^{(5)}$
EVAL
EVAL
$\mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}}) \nsim \mathsf{app}([\,], XS, XS)$
J $\square_{\sigma_6\sigma_7} \mid \mathsf{app}([\,], T_{11}, \overline{T_{13}})_{\sigma_6\sigma_8}^{(5)}$
$\mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}})_{\sigma_6}^{(5)}$
SUC
$\mathsf{app}(\overline{T_{10}}, T_{11}, \overline{T_{12}}) \nsim \mathsf{app}([X \mid XS], YS, [X \mid ZS])$
EVAL
EVAL
$\mathsf{app}([\,], T_{11}, \overline{T_{13}})_{\sigma_6\sigma_8}^{(5)}$
ε
BACKTRACK
K $\mathsf{app}(\overline{T_{15}}, T_{16}, \overline{T_{17}})_{\sigma_6\sigma_9}$
ε

Figure 6: Symbolic Evaluation Graph for Ex. 2

$(\mathsf{star}(T_1, T_2)_{id}; (\{T_1, T_2\}, \varnothing))$. A *symbolic evaluation* starting with this state A is depicted in Fig. 6. The nodes of such a symbolic evaluation graph are states and each step from a node to its children is done by an inference rule. To save space, we omitted the knowledge base from the states $(S; (\mathcal{G}, \mathcal{U}))$. Instead, we overlined all variables contained in $\mathcal{G}$ and labeled those edges where new information is added to $\mathcal{U}$.

The child of A is B with $(\mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(1')} \mid \mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(2')} \mid \mathsf{star}(\overline{T_1}, \overline{T_2})_{id}^{(3)} \mid ?_1)$. In Fig. 6 we simplified the states by removing markers $?_m$ that occur at the end of a state. This is possible, since applying the first CUT rule to a state ending in $?_m$ corresponds to applying the second CUT rule to the same state without $?_m$. Moreover, $(1')$ and $(2')$ again abbreviate $(1)[!/!_1]$ and $(2)[!/!_1]$.

In B, $(1')$ is used for the next evaluation. EVAL yields two successors: In C, $\sigma_1 = mgu(\mathsf{star}(\overline{T_1}, \overline{T_2}), \mathsf{star}(XS, [\,])) = \{\overline{T_1}/\overline{T_3}, XS/\overline{T_3}, \overline{T_2}/[\,]\}$ leads to $((!_1)_{\sigma_1} \mid \mathsf{star}(\overline{T_3}, [\,])_{\sigma_2}^{(2')} \mid \mathsf{star}(\overline{T_3}, [\,])_{\sigma_2}^{(3)})$. Here, $\sigma_2 = \sigma_1|_{\{\overline{T_1}, \overline{T_2}\}}$. In the second successor D of B, we add the information $\mathsf{star}(\overline{T_1}, \overline{T_2}) \nsim \mathsf{star}(XS, [\,])$ to $\mathcal{U}$ (thus, we labeled the edge from B to D accordingly).

Unfortunately, even for terminating queries, in general the rules of Def. 4 yield an infinite tree. The reason is that there is no bound on the size of terms represented by the abstract variables and hence, the abstract Eval rule can be applied infinitely often. To represent all possible evaluations in a finite way, we need additional inference rules to obtain finite symbolic evaluation graphs instead of infinite trees.

To this end, we use an additional Inst rule which allows us to connect the current state $(S; KB)$ with a previous state $(S'; KB')$, provided that the current state is an instance of the previous state. In other words, every concretization of $(S; KB)$ must be a concretization of $(S'; KB')$. More precisely, there must be a matching substitution μ such that $S'\mu = S$ up to the substitutions used for labeling goals in S' and S. These substitution labels do not have to be taken into account here, since we will not generate rewrite rules from paths that traverse Inst edges in Sect. 4. Moreover, for $KB' = (\mathcal{G}',\mathcal{U}')$ and $KB = (\mathcal{G},\mathcal{U})$, $\mathcal{G}'$ and $\mathcal{G}$ must be the same (modulo μ) and all constraints from $\mathcal{U}'$ must occur in $\mathcal{U}$ (modulo μ). Then we say that μ is *associated* to $(S; KB)$ and label the resulting Inst edge with μ. For example, in Fig. 6, $\mu = \{\overline{T_1}/\overline{T_5}, \overline{T_2}/\overline{T_8}\}$ is associated to H and the edge from H to A is labeled with μ. We only define the Inst rule for states containing a single goal. As indicated by our experiments, this is no severe restriction in practice.[3]

Definition 7 (Abstract Rules: Inst).

$$\frac{S; (\mathcal{G},\mathcal{U})}{S'; (\mathcal{G}',\mathcal{U}')}\ (\textsc{Inst})$$

if $S = Q_\theta$ and $S' = Q'_{\theta'}$ or $S = Q^c_\theta$ and $S' = Q'^c_{\theta'}$ for some non-empty queries Q and Q', such that there is a μ with $Dom(\mu) \subseteq \mathcal{A}$, $\mathcal{V}(Range(\mu)) \subseteq \mathcal{A}$, $Q = Q'\mu$, $\mathcal{G} = \bigcup_{T\in\mathcal{G}'} \mathcal{V}(T\mu)$, and $\mathcal{U}'\mu \subseteq \mathcal{U}$.

Thm. 8 states that every concrete state represented by an Inst node is also represented by its successor.

Theorem 8 (Soundness of Inst). *Let $(S; KB)$ be an abstract state, let $(S'; KB')$ be its successor according to the* Inst *rule, and let μ be associated to $(S; KB)$. If $S\gamma \in \mathcal{CON}(S;KB)$, then for $\gamma'{=}\mu\gamma$ we have $S'\gamma' \in \mathcal{CON}(S';KB')$.*

Moreover, we also need a Split inference rule to split a state $((t, Q)_\theta; KB)$ into $(t_{id}; KB)$ and $((Q\delta)_\delta; KB')$, where δ approximates the answer substitutions for t. Such a Split is often needed to make the Inst rule applicable. We say that δ is *associated* to $((t, Q)_\theta; KB)$. The previous substitution label θ does not have to be taken into account here, since we will not generate rewrite rules from paths that traverse Split nodes in Sect. 4. Thus, we can reset the substitution label θ to *id* in the first successor of the Split node and store the associated substitution δ in the substitution label of the second successor. Similar to the Inst rule, we only define the Split rule for states containing a single goal.

[3]In [36] and in our implementation, we use an additional inference rule to split up sequences of goals, but we omitted it here for readability. Adding this rule allows us to construct a symbolic evaluation graph for each program and query.

Definition 9 (Abstract Rules: Split).

$$\frac{(t, Q)_\theta; (\mathcal{G},\mathcal{U})}{t_{id}; (\mathcal{G},\mathcal{U}) \quad (Q\delta)_\delta; (\mathcal{G}',\mathcal{U}\delta)}\ (\textsc{Split})$$

where δ replaces all previously occurring variables from $\mathcal{A} \setminus \mathcal{G}$ by fresh abstract variables and $\mathcal{G}' = \mathcal{G} \cup NextG(t, \mathcal{G})\delta$.

Here, *NextG* is defined as follows. We assume that we have a *groundness analysis* function $Ground_\mathcal{P} : \Sigma \times 2^\mathbb{N} \to 2^\mathbb{N}$, see, e.g., [21]. If $\mathsf{p}/n \in \Sigma$ and $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$, then $Ground_\mathcal{P}(\mathsf{p}, \{i_1, \ldots, i_m\}) {=} \{j_1, \ldots, j_k\}$ means that any query $\mathsf{p}(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{N})$ where $t_{i_1}, \ldots, t_{i_m}$ are ground only has answer substitutions θ where $t_{j_1}\theta, \ldots, t_{j_k}\theta$ are ground. So $Ground_\mathcal{P}$ approximates which positions of p will become ground if the "input" positions $i_1, \ldots, i_m$ are ground. Now if $t = \mathsf{p}(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{A})$ is an abstract term where $t_{i_1}, \ldots, t_{i_m}$ become ground in every concretization (i.e., all their variables are from $\mathcal{G}$), then $NextG(t, \mathcal{G})$ returns all variables in t that will be made ground by every answer substitution for any concretization of t. Thus, $NextG(t, \mathcal{G})$ contains the variables of $t_{j_1}, \ldots, t_{j_k}$. So formally

$$NextG(\mathsf{p}(t_1, \ldots, t_n), \mathcal{G}) = \bigcup\nolimits_{j \in Ground_\mathcal{P}(\mathsf{p},\ \{i \mid \mathcal{V}(t_i) \subseteq \mathcal{G}\}\ )} \mathcal{V}(t_j).$$

Hence, in the second successor of the Split rule, the variables in $NextG(t, \mathcal{G})$ can be added to the groundness set $\mathcal{G}$. Since these variables were renamed by δ, we extend $\mathcal{G}$ by $NextG(t, \mathcal{G})\delta$.

For instance, in Fig. 6, we split the query $\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})$, $\mathsf{star}(\overline{T_5}, T_7)$ in state F. Thus, the first successor of F is $\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})$ in state G. By groundness analysis, we infer that every successful evaluation of $\mathsf{app}(\overline{T_5}, T_7, \overline{T_6})$ instantiates T_7 by ground terms, i.e., $Ground_\mathcal{P}(\mathsf{app}, \{1, 3\}) = \{1, 2, 3\}$. Thus, for $\mathcal{G} = \{T_5, T_6\}$, we have $NextG(\mathsf{app}(T_5, T_7, T_6), \mathcal{G}) = \mathcal{V}(T_5) \cup \mathcal{V}(T_7) \cup \mathcal{V}(T_6) = \{T_5, T_7, T_6\}$. So in the second successor H of F, we use the substitution $\delta(T_7) = T_8$ and extend the groundness set $\mathcal{G}$ of F by $NextG(\mathsf{app}(T_5, T_7, T_6), \mathcal{G})\delta = \{T_5, T_8, T_6\}$. Thus, T_8 is also overlined in Fig. 6.

Thm. 10 shows the soundness of Split. Suppose that we apply the Split rule to $((t, Q)_\theta; KB)$, which yields $(t_{id}; KB)$ and $((Q\delta)_\delta; KB')$. Any evaluation of a concrete state $(t\gamma, Q\gamma) \in \mathcal{CON}((t, Q)_\theta; KB)$ consists of parts where one evaluates $t\gamma$ (yielding some answer substitution θ') and of parts where one evaluates $Q\gamma\theta'$. Clearly, those parts which correspond to evaluations of $t\gamma$ can be simulated by the left successor of the Split node (since $t\gamma \in \mathcal{CON}(t_{id}; KB)$). Thm. 10 states that the parts of the overall evaluation which correspond to evaluations of $Q\gamma\theta'$ can be simulated by the right successor of the Split node (i.e., $Q\gamma\theta' \in \mathcal{CON}((Q\delta)_\delta; KB')$).

Theorem 10 (Soundness of Split). *Let $((t,Q)_\theta;KB)$ be an abstract state and let $(t_{id}; KB)$ and $((Q\delta)_\delta; KB')$ be its successors according to the* Split *rule. Let $(t\gamma, Q\gamma) \in \mathcal{CON}((t, Q)_\theta; KB)$ and let θ' be an answer substitution of $(t\gamma)_{id}$. Then we have $Q\gamma\theta' \in \mathcal{CON}((Q\delta)_\delta; KB')$.*

We define *symbolic evaluation graphs* as a subclass of the graphs obtained by the rules of Def. 4, 7, and 9. They must not have any cycles consisting only of Inst edges, as this would lead to trivially non-terminating TRSs. Moreover, their only leaves may be nodes where no inference rule is applicable anymore (i.e., the graphs must be "fully expanded"). The graph in Fig. 6 is indeed a symbolic evaluation graph.

Definition 11 (Symbolic Evaluation Graph). *A finite graph built from an initial state using Def. 4, 7, and 9 is a* symbolic evaluation graph *(or "evaluation graph" for short) iff there is no cycle consisting only of* Inst *edges and all leaves are of the form $(\varepsilon; KB)$.*[4]

4. FROM EVALUATION GRAPHS TO TRSS – TERMINATION ANALYSIS

Now our goal is to show termination of all concrete states represented by the graph's initial state. To this end, we synthesize a TRS from the symbolic evaluation graph. This TRS has the following property: if there is an evaluation from a concretization of one state to a concretization of another state which may be crucial for termination, then there is a corresponding rewrite sequence w.r.t. the TRS. Then automated tools for termination analysis of TRSs can be used to show termination of the synthesized TRS and this implies termination of the original logic program. See, e.g., [13, 16, 42] for an overview of techniques for automatically proving termination of TRSs.

For the basics of term rewriting, we refer to [6]. A *term rewrite system* $\mathcal{R}$ is a finite set of rules $\ell \to r$ where $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$. The rewrite relation $t \to_{\mathcal{R}} t'$ for two terms t and t' holds iff there is an $\ell \to r \in \mathcal{R}$, a position *pos*, and a substitution σ such that $\ell\sigma = t|_{pos}$ and $t' = t[r\sigma]_{pos}$. Here, $t|_{pos}$ is the subterm of t at position *pos* and $t[r\sigma]_{pos}$ results from replacing the subterm $t|_{pos}$ at position *pos* in t by the term $r\sigma$. The rewrite step is *innermost* (denoted $t \xrightarrow{\text{i}}_{\mathcal{R}} t'$) iff no proper subterm of $\ell\sigma$ can be rewritten.

To obtain a TRS from an evaluation graph *Gr*, we encode the states as terms. For each state $s = (S; (\mathcal{G}, \mathcal{U}))$, we use two fresh function symbols f_s^{in} and f_s^{out}. The arguments of f_s^{in} are the variables in $\mathcal{G}$ (which represent ground terms). The arguments of f_s^{out} are those remaining abstract variables which will be made ground by every answer substitution for any concretization of s. They are again determined by groundness analysis [21]. Formally, the encoding of states is done by two functions enc^{in} and enc^{out}.

For instance, for the state F in Fig. 6, we obtain $enc^{in}(\text{F}) = f_{\text{F}}^{in}(T_5, T_6)$ (as $\mathcal{G} = \{T_5, T_6\}$ in F) and $enc^{out}(\text{F}) = f_{\text{F}}^{out}(T_7)$. The reason is that if γ instantiates T_5 and T_6 by ground terms, then every answer substitution of $(\mathsf{app}(T_5, T_7, T_6)\gamma$, $\mathsf{star}(T_5, T_7)\gamma)$ instantiates $T_7\gamma$ to a ground term as well.

For an Inst node like H with associated substitution μ we do not introduce fresh function symbols, but use the function symbol of its (more general) successor instead. So we take the terms resulting from its successor A and apply μ to them. In other words, $enc^{in}(\text{H}) = enc^{in}(\text{A})\,\mu = f_{\text{A}}^{in}(T_1, T_2)\,\mu = f_{\text{A}}^{in}(T_5, T_8)$ and $enc^{out}(\text{H}) = enc^{out}(\text{A})\,\mu = f_{\text{A}}^{out}\,\mu = f_{\text{A}}^{out}$.

In the following, for an evaluation graph *Gr* and an inference rule Rule, *Rule*(*Gr*) denotes all nodes of *Gr* to which Rule was applied. Let $Succ_i(s)$ denote the i-th child of node s and $Succ_i(Rule(Gr))$ denotes the set of i-th children of all nodes from *Rule*(*Gr*).

Definition 12 (Encoding States as Terms). *Let s be an abstract state with a single goal (i.e., $s = ((t_1, \ldots, t_k)_\theta; (\mathcal{G}, \mathcal{U}))$), and let $\mathcal{V}(s) = \mathcal{V}(t_1) \cup \ldots \cup \mathcal{V}(t_k)$. We define*

$$enc^{in}(s) = \begin{cases} enc^{in}(Succ_1(s))\,\mu, & \textit{if } s \in Inst(Gr) \textit{ where } \mu \textit{ is associated to } s \\ f_s^{in}(\mathcal{G}^{in}(s)), & \textit{otherwise, where } \mathcal{G}^{in}(s) = \mathcal{G} \cap \mathcal{V}(s) \end{cases}$$

$$enc^{out}(s) = \begin{cases} enc^{out}(Succ_1(s))\,\mu, & \textit{if } s \in Inst(Gr) \textit{ where } \mu \textit{ is associated to } s \\ f_s^{out}(\mathcal{G}^{out}(s)), & \textit{otherwise, where } \mathcal{G}^{out}(s) = NextG((t_1, \ldots, t_k), \mathcal{G}) \setminus \mathcal{G} \end{cases}$$

Here, we extended NextG to work also on queries:

$$\begin{aligned} NextG((t_1, \ldots, t_k), \mathcal{G}) \;=\; & NextG(t_1, \mathcal{G}) \cup \\ & NextG(\,(t_2, \ldots, t_k),\, NextG(t_1, \mathcal{G})\,). \end{aligned}$$

So to compute $NextG((t_1, \ldots, t_k), \mathcal{G})$ for a query $(t_1, \ldots, t_k)$, in the beginning we only know that the variables in $\mathcal{G}$ represent ground terms. Then we compute the variables $NextG(t_1, \mathcal{G})$ which are made ground by all answer substitutions for concretizations of t_1. Next, we compute $NextG(t_2, NextG(t_1, \mathcal{G}))$ which are made ground by all answer substitutions for concretizations of t_2, etc.

Now we encode the paths of *Gr* as rewrite rules. However, we only consider *connection paths* of *Gr*, which suffice to analyze termination. Connection paths are non-empty paths that start in the root node of the graph or in a successor of an Inst or Split node, provided that these states are not Inst or Split nodes themselves. So the start states in our example are A, G, and I. Moreover, connection paths end in an Inst, Split, or Suc node or in the successor of an Inst node, while not traversing Inst or Split nodes or successors of Inst nodes in between. So in our example, the end states are A, E, F, H, I, J, K, but apart from E and J, connection paths may not traverse any of these end nodes in between.

Thus, we have connection paths from A to E, A to F, G to I, I to J, and I to K. These paths cover all ways through the graph except for Inst edges (which are covered by the encoding of states to terms), for Split edges (which we consider later in Def. 15), and for graph parts without cycles or Suc nodes (which cannot cause non-termination).

Definition 13 (Connection Path). *A path $\pi = s_1 ... s_k$ is a* connection path *of an evaluation graph Gr iff $k > 1$ and*

- $s_1 \in \{root(Gr)\} \cup Succ_1(Inst(Gr) \cup Split(Gr)) \cup Succ_2(Split(Gr))$
- $s_k \in Inst(Gr) \cup Split(Gr) \cup Suc(Gr) \cup Succ_1(Inst(Gr))$
- *for all* $1 \leq j < k$, $s_j \notin Inst(Gr) \cup Split(Gr)$
- *for all* $1 < j < k$, $s_j \notin Succ_1(Inst(Gr))$

For a connection path π, let σ_π represent the unifiers that were applied along the path. These unifiers can be determined by "comparing" the substitution labels of the first and the last state of the path (i.e., the goal in π's first state has a substitution label θ and the first goal of π's last state is labeled by $\theta\sigma_\pi$). So for the connection path π from A to F we have $\sigma_\pi = \sigma_5$, where $\sigma_5(\overline{T_1}) = \overline{T_5}$ and $\sigma_5(\overline{T_2}) = \overline{T_6}$. For this path, we generate rewrite rules which evaluate the instantiated input term $enc^{in}(\text{A})\,\sigma_\pi$ for the start node A to its output term $enc^{out}(\text{A})\,\sigma_\pi$ if the input term $enc^{in}(\text{F})$ for the end node can be evaluated to its output term $enc^{out}(\text{F})$. So we get $enc^{in}(\text{A})\,\sigma_\pi \to \mathsf{u}_{\text{A,F}}(\,enc^{in}(\text{F}), \mathcal{V}(enc^{in}(\text{A})\,\sigma_\pi)\,)$ and $\mathsf{u}_{\text{A,F}}(\,enc^{out}(\text{F}),\, \mathcal{V}(enc^{in}(\text{A})\,\sigma_\pi)\,) \to enc^{out}(\text{A})\,\sigma_\pi$ for a fresh function symbol $\mathsf{u}_{\text{A,F}}$. In our example, this yields

$$f_{\text{A}}^{in}(T_5, T_6) \to \mathsf{u}_{\text{A,F}}(f_{\text{F}}^{in}(T_5, T_6), T_5, T_6) \qquad (7)$$

$$\mathsf{u}_{\text{A,F}}(f_{\text{F}}^{out}(T_7), T_5, T_6) \to f_{\text{A}}^{out} \qquad (8)$$

[4]The application of inference rules to abstract states is not deterministic. In our prover AProVE, we implemented a heuristic [38] to generate symbolic evaluation graphs automatically which turned out to be very suitable for subsequent analyses in our empirical evaluations.

However, for connection paths π' like the one from A to E which end in a SUC node, the resulting rewrite rule directly evaluates the instantiated input term $enc^{in}(\text{A})\,\sigma_{\pi'}$ for the start node A to its output term $enc^{out}(\text{A})\,\sigma_{\pi'}$. So we obtain

$$f^{in}_{\text{A}}(T_3, [\,]) \to f^{out}_{\text{A}} \tag{9}$$

DEFINITION 14 (RULES FOR CONNECTION PATHS). *Let π be a connection path $s_1 \dots s_k$ in a symbolic evaluation graph. Let the (only) goal in s_1 be labeled by the substitution θ and let the first goal in s_k be labeled by the substitution $\theta\,\sigma_\pi$. If $s_k \in Suc(Gr)$, then we define $ConnectionRules(\pi) =$*

$$\{enc^{in}(s_1)\sigma_\pi \to enc^{out}(s_1)\,\sigma_\pi\}.$$

Otherwise, $ConnectionRules(\pi)$ =

$$\begin{aligned}\{\, & enc^{in}(s_1)\,\sigma_\pi \;\to\; \mathsf{u}_{s_1,s_k}(\, enc^{in}(s_k),\, \mathcal{V}(enc^{in}(s_1)\,\sigma_\pi)\,),\\ & \mathsf{u}_{s_1,s_k}(\, enc^{out}(s_k),\, \mathcal{V}(enc^{in}(s_1)\,\sigma_\pi)\,) \;\to\; enc^{out}(s_1)\,\sigma_\pi \,\},\end{aligned}$$

where u_{s_1,s_k} is a fresh function symbol.

In addition to the rules for connection paths, we also need rewrite rules to simulate the evaluation of SPLIT nodes like F. Let δ be the substitution associated to F (i.e., δ represents the answer substitution of F's first successor G). Then the SPLIT node F succeeds (i.e., $enc^{in}(\text{F})\,\delta$ can be evaluated to $enc^{out}(\text{F})\,\delta$) if both successors G and H succeed (i.e., $enc^{in}(\text{G})\,\delta$ can be evaluated to $enc^{out}(\text{G})\,\delta$ and $enc^{in}(\text{H})$ can be evaluated to $enc^{out}(\text{H})$). Note that $enc^{in}(\text{F})$ and $enc^{in}(\text{G})$ only contain "input" arguments (i.e., abstract variables from $\mathcal{G}$) and thus, δ does not modify them. Hence, $enc^{in}(\text{F})\,\delta = enc^{in}(\text{F})$ and $enc^{in}(\text{G})\,\delta = enc^{in}(\text{G})$. So we obtain

$$f^{in}_{\text{F}}(T_5, T_6) \to \mathsf{u}_{\text{F,G}}(f^{in}_{\text{G}}(T_5, T_6), T_5, T_6) \tag{10}$$

$$\mathsf{u}_{\text{F,G}}(f^{out}_{\text{G}}(T_8), T_5, T_6) \to \mathsf{u}_{\text{G,H}}(f^{in}_{\text{A}}(T_5, T_8), T_5, T_6, T_8) \tag{11}$$

$$\mathsf{u}_{\text{G,H}}(f^{out}_{\text{A}}, T_5, T_6, T_8) \to f^{out}_{\text{F}}(T_8) \tag{12}$$

DEFINITION 15 (RULES FOR SPLIT, $\mathcal{R}(Gr)$). *Let $s \in Split(Gr)$, $s_1 = Succ_1(s)$, and $s_2 = Succ_2(s)$. Moreover, let δ be the substitution associated to s. Then $SplitRules(s)$ =*

$$\begin{aligned}\{\, & enc^{in}(s) \;\to\; \mathsf{u}_{s,s_1}(\, enc^{in}(s_1),\, \mathcal{V}(enc^{in}(s))\,),\\ & \mathsf{u}_{s,s_1}(\, enc^{out}(s_1)\,\delta,\, \mathcal{V}(enc^{in}(s))\,) \;\to\\ & \qquad\qquad \mathsf{u}_{s_1,s_2}(\, enc^{in}(s_2),\, \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)\,),\\ & \mathsf{u}_{s_1,s_2}(enc^{out}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\,\delta)) \to enc^{out}(s)\delta\}\end{aligned}$$

$\mathcal{R}(Gr)$ consists of $ConnectionRules(\pi)$ for all connection paths π and of $SplitRules(s)$ for all SPLIT *nodes s of Gr.*

For the graph Gr of Fig. 6, the resulting TRS $\mathcal{R}(Gr)$ consists of (7) – (12) and the connection rules (13), (14) for the path from G to I (where $\sigma_6(\overline{T_5}) = [\overline{T_9}\,|\,\overline{T_{10}}], \sigma_6(T_7) = T_{11}, \sigma_6(\overline{T_6}) = [\overline{T_9}\,|\,\overline{T_{12}}]$), the rules (15), (16) for I to K (where $\sigma_9(\overline{T_{10}}) = [\overline{T_{14}} \mid \overline{T_{15}}], \sigma_9(T_{11}) = T_{16}, \sigma_9(\overline{T_{12}}) = [\overline{T_{14}} \mid \overline{T_{17}}]$), and (17) for I to J (where $\sigma_8 = \sigma_7|_{\{\overline{T_{10}},\overline{T_{12}}\}}$ with $\sigma_8(\overline{T_{10}}) = [\,], \sigma_8(\overline{T_{12}}) = \overline{T_{13}}$).

$$f^{in}_{\text{G}}([T_9 \mid T_{10}], [T_9 \mid T_{12}]) \to \mathsf{u}_{\text{G,I}}(f^{in}_{\text{I}}(T_{10}, T_{12}), T_9, T_{10}, T_{12}) \tag{13}$$

$$\mathsf{u}_{\text{G,I}}(f^{out}_{\text{I}}(T_{11}), T_9, T_{10}, T_{12}) \to f^{out}_{\text{G}}(T_{11}) \tag{14}$$

$$f^{in}_{\text{I}}([T_{14} \mid T_{15}], [T_{14} \mid T_{17}]) \to \mathsf{u}_{\text{I,K}}(f^{in}_{\text{I}}(T_{15}, T_{17}), T_{14}, T_{15}, T_{17}) \tag{15}$$

$$\mathsf{u}_{\text{I,K}}(f^{out}_{\text{I}}(T_{16}), T_{14}, T_{15}, T_{17}) \to f^{out}_{\text{I}}(T_{16}) \tag{16}$$

$$f^{in}_{\text{I}}([\,], T_{13}) \to f^{out}_{\text{I}}(T_{13}) \tag{17}$$

Thm. 16 states that the resulting TRS can simulate all successful evaluations represented in the graph, i.e., it simulates all computations of the logic program.

THEOREM 16 (TRS SIMULATES SEMANTICS). *Let $s = (S; KB)$ be a start node of a connection path or a* SPLIT *node in a graph Gr, $S\gamma \in \mathcal{CON}(s)$, and let θ be an answer substitution for $S\gamma$. Then $enc^{in}(s)\gamma \xrightarrow{\text{i}}{}^{+}_{\mathcal{R}(Gr)} enc^{out}(s)\gamma\theta$.*

Virtually all modern TRS termination tools can prove that $\mathcal{R}(Gr)$ is terminating in our example. Thm. 17 shows that this implies termination of all queries corresponding to the root of Gr. Hence, by our approach, one can prove termination of non-definite logic programs like Ex. 2 automatically.

THEOREM 17 (SOUNDNESS OF TERMINATION ANALYSIS). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}^{\mathsf{p}}_m$. If the TRS $\mathcal{R}(Gr)$ is innermost terminating, then there is no infinite evaluation starting with any query from $\mathcal{Q}^{\mathsf{p}}_m$. Thus, all these queries are terminating w.r.t. the program $\mathcal{P}$.*

We implemented our approach for termination analysis in the tool AProVE [15]. In addition to the cut, our implementation handles many further features of Prolog. For our experiments, AProVE ran on all 477 Prolog programs of the *Termination Problem Database* (TPDB, version 8.0.6), which is the collection of examples used in the annual *International Termination Competition.*[5] 300 of them are definite logic programs, whereas the remaining 177 programs contain advanced features like cuts. 37 of the 477 examples are known to be non-terminating. The experiments were run on 2.2 GHz Quad-Opteron 848 Linux machines with a timeout of 60 seconds per program. In the table, "**Yes**" indicates the number of examples where termination could be proved and "**RT**" is the average runtime (in seconds) per example.

	Yes	**RT**
AProVE-[34]	265	7.1
AProVE-[36]	287	7.6
AProVE-[39]	340	5.7
AProVE-New	342	6.5

All termination tools for logic programs except AProVE ignore cuts, i.e., they try to prove termination of the program that results from removing the cuts. This is sensible, since cuts are not always needed for termination. Indeed, the variant AProVE-[34] implements our technique from [34] which ignores cuts and directly translates logic programs to TRSs. Still, it proves termination of 31 of the 177 non-definite programs. Other existing termination tools would not yield better results, as AProVE-[34] is already the most powerful tool for definite logic programs (as shown by the experiments in [34]) and as most of the remaining non-definite examples do not terminate anymore if one removes cuts. AProVE-[36] implements our approach from [36] which introduced evaluation graphs, but transforms them to definite logic programs instead of TRSs. This approach is much more powerful than [34] on examples with cut, but it fails on many definite logic programs where [34] was successful. The approach of the current paper (implemented in AProVE-New)[6] considers other paths in the graph than [36]. Thus, it

[5]In these competitions, AProVE was the most powerful tool for termination of logic programs, see `http://termination-portal.org/wiki/Termination_Competition/`.

[6]To benefit from the full power of rewriting-based termination analysis, in our implementation we generate TRSs together with an *argument filtering*, as in [34]. In this way, one can also handle examples where ground information on the arguments of predicates is not sufficient.

simulates the evaluations of the original logic program more concisely and results in a more powerful approach (both for definite and non-definite programs).

[39] improved upon [36] by generating "dependency triples" from evaluation graphs. Indeed, AProVE-New and AProVE-[39] have almost the same power. But while the back-end of [39] required a tool that can handle the (non-standard) notion of dependency triples, our new approach works with any tool for termination of TRSs. Moreover, the approach of the current paper has the advantage that the TRSs generated for termination analysis can also be used for analyzing other properties like complexity, as shown in Sect. 5.

5. FROM EVALUATION GRAPHS TO TRSS – COMPLEXITY ANALYSIS

We briefly recapitulate the required notions for complexity of TRSs. The *defined symbols* of a TRS $\mathcal{R}$ are $\Sigma_d = \{root(\ell) \mid \ell \to r \in \mathcal{R}\}$, i.e., these are the function symbols that can be "evaluated". So for $\mathcal{R}(Gr)$ from Sect. 4, we have $\Sigma_d = \{f^{in}_{\mathrm{A}}, \mathsf{u}_{\mathrm{A,F}}, f^{in}_{\mathrm{F}}, \mathsf{u}_{\mathrm{F,G}}, \mathsf{u}_{\mathrm{G,H}}, f^{in}_{\mathrm{G}}, \mathsf{u}_{\mathrm{G,I}}, f^{in}_{\mathrm{I}}, \mathsf{u}_{\mathrm{I,K}}\}$. Different notions of complexity have been proposed for TRSs.In this paper, we focus on *innermost runtime complexity* [20], which corresponds to the notion of complexity used for programming languages. Here, one only considers rewrite sequences starting with *basic* terms $\mathsf{f}(t_1, \dots, t_n)$, where $\mathsf{f} \in \Sigma_d$ and $t_1, \dots, t_n$ do not contain symbols from Σ_d. The *innermost runtime complexity function* $irc_{\mathcal{R}}$ maps any $n \in \mathbb{N}$ to the length of the longest sequence of $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$-steps starting with a basic term t where $|t| \leq n$. Here, $|t|$ is the number of variables and function symbols occurring in t. To measure the complexity of a TRS $\mathcal{R}$, we determine the asymptotic growth of $irc_{\mathcal{R}}$, i.e., we say that $\mathcal{R}$ has linear complexity iff $irc_{\mathcal{R}}(n) \in \mathcal{O}(n)$, quadratic complexity iff $irc_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$, etc. Tools for automated complexity analysis of TRSs can automatically determine $irc_{\mathcal{R}(Gr)}(n) \in \mathcal{O}(n)$ for $\mathcal{R}(Gr) = \{(7) - (17)\}$ from Sect. 4.[7]

Moreover, we also have to define the notion of "complexity" for a logic programs. For a logic program $\mathcal{P}$ and a query Q, we consider the length of the longest evaluation starting in the initial state for Q. As shown in [40], this length is equal to the number of unification attempts when traversing the whole SLD tree according to the ISO semantics [22], up to a constant factor.[8] For a moding function m, and any term $\mathsf{p}(t_1, \dots, t_n)$, its *moded size* is $|\mathsf{p}(t_1, \dots, t_n)|_m = 1 + \Sigma_{i \in \{i \mid 1 \leq i \leq n, m(\mathsf{p},i)=in\}} |t_i|$. Thus, for a class of queries $\mathcal{Q}^{\mathsf{p}}_m$, the Prolog *runtime complexity function* $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}$ maps any $n \in \mathbb{N}$ to the length of the longest evaluation starting with the initial state for some query $Q \in \mathcal{Q}^{\mathsf{p}}_m$ with $|Q|_m \leq n$.

To analyze $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n)$, we generate an evaluation graph Gr for $\mathcal{Q}^{\mathsf{p}}_m$ as in Sect. 3 and obtain the TRS $\mathcal{R}(Gr)$ as in Sect. 4. At first sight, one might expect that asymptotically, $irc_{\mathcal{R}(Gr)}(n)$ is indeed an upper bound of $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n)$. This would allow us to use existing methods for complexity analysis of TRSs in order to derive upper bounds on the runtime of logic programs.

In fact for Ex. 2, both $irc_{\mathcal{R}(Gr)}(n)$ and $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{star}}_m}(n)$ are in $\mathcal{O}(n)$, i.e., the complexity of the logic program for $\mathcal{Q}^{\mathsf{star}}_m$ is also linear. But in general, $irc_{\mathcal{R}(Gr)}(n)$ is not necessarily an upper bound of $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n)$. This can happen if Gr contains a SPLIT node whose first successor is *not deterministic*. A query Q is *deterministic* iff it generates at most one answer substitution at most once [24]. Similarly, we call an abstract state s *deterministic* iff each of its concretizations has at most one evaluation to a state of the form $(\square_\theta \mid S)$.

[7]For example, this can be determined by the tool TCT [3]. While AProVE was the most powerful tool for innermost runtime complexity analysis in the recent termination competitions, here it only obtains $irc_{\mathcal{R}(Gr)}(n) \in \mathcal{O}(n^2)$.

[8]In contrast, other approaches like [10, 11, 12, 29] use the number of resolution steps to measure complexity. As long as one does not consider dynamic built-in predicates like assert/1, these measures are asymptotically equivalent, as the number of unification attempts at each resolution step is bounded by a constant (i.e., by the number of program clauses).

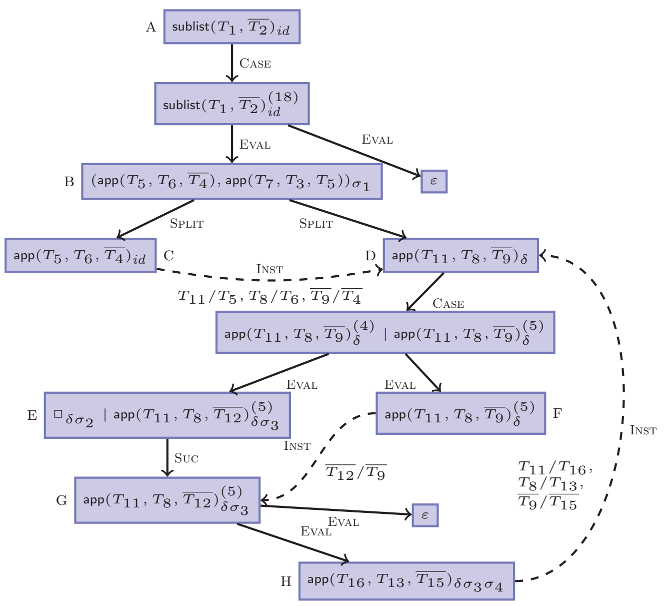

A
sublist$(T_1, \overline{T_2})_{id}$
CASE
sublist$(T_1, \overline{T_2})^{(18)}_{id}$
EVAL
EVAL
ε
B
$(\mathsf{app}(T_5, T_6, \overline{T_4}), \mathsf{app}(T_7, T_3, T_5))_{\sigma_1}$
SPLIT
SPLIT
app$(T_5, T_6, \overline{T_4})_{id}$
C
INST
D
app$(T_{11}, T_8, \overline{T_9})_{\delta}$
$T_{11}/T_5, T_8/T_6, \overline{T_9}/\overline{T_4}$
CASE
app$(T_{11}, T_8, \overline{T_9})^{(4)}_{\delta}$ | app$(T_{11}, T_8, \overline{T_9})^{(5)}_{\delta}$
EVAL
EVAL
E
$\square_{\delta\sigma_2}$ | app$(T_{11}, T_8, \overline{T_{12}})^{(5)}_{\delta\sigma_3}$
app$(T_{11}, T_8, \overline{T_9})^{(5)}_{\delta}$
F
INST
SUC
INST
$\overline{T_{12}}/\overline{T_9}$
T_{11}/T_{16}, T_8/T_{13}, $\overline{T_9}/\overline{T_{15}}$
G
app$(T_{11}, T_8, \overline{T_{12}})^{(5)}_{\delta\sigma_3}$
EVAL
EVAL
ε
H
app$(T_{16}, T_{13}, \overline{T_{15}})_{\delta\sigma_3\sigma_4}$

Figure 19: Symbolic Evaluation Graph for Ex. 18

EXAMPLE 18. *To see the problems with* SPLIT *nodes whose first successor is not deterministic, consider the following program from the TPDB which consists of the clauses (4) and (5) for* app *and the following rule:*

$$\mathsf{sublist}(X, Y) \text{ :- } \mathsf{app}(P, U, Y),\ \mathsf{app}(V, X, P). \qquad (18)$$

We regard the class of queries $\mathcal{Q}^{\mathsf{sublist}}_m$, where $m(\mathsf{sublist}, 1) = out$ and $m(\mathsf{sublist}, 2) = in$. The program computes (by backtracking) all sublists of a given list. Its complexity is quadratic since the first app*-call results in a linear evaluation with a linear number of solutions. The second* app*-call again needs linear time, but due to backtracking, it is called linearly often.*

We obtain the evaluation graph Gr in Fig. 19. For readability, we omitted labels $t \nsim t'$ on EVAL*-edges. We have $\sigma_1(T_1) = T_3, \sigma_1(\overline{T_2}) = \overline{T_4}$; $\sigma_2(T_8) = \overline{T_{12}}, \sigma_2(\overline{T_9}) = \overline{T_{12}}$, $\sigma_2(T_{11}) = [\,]$; $\sigma_3(\overline{T_9}) = \overline{T_{12}}$; $\sigma_4(T_8) = T_{13}, \sigma_4(\overline{T_{12}}) = [\overline{T_{14}} \mid \overline{T_{15}}], \sigma_4(T_{11}) = [T_{14} \mid T_{16}]$; and $\delta(T_3) = T_8, \delta(T_5) = \overline{T_9}$, $\delta(T_6) = \overline{T_{10}}, \delta(T_7) = T_{11}$.*

This symbolic evaluation graph has connection paths from A *to* B, D *to* E, D *to* G, D *to* F, *and* G *to* H. *It gives rise to the following TRS $\mathcal{R}(Gr)$.*

$$f^{in}_{\mathrm{A}}(T_4) \to \mathsf{u}_{\mathrm{A,B}}(f^{in}_{\mathrm{B}}(T_4), T_4) \tag{19}$$
$$\mathsf{u}_{\mathrm{A,B}}(f^{out}_{\mathrm{B}}(T_5, T_6, T_7, T_3), T_4) \to f^{out}_{\mathrm{A}}(T_3) \tag{20}$$
$$f^{in}_{\mathrm{B}}(T_4) \to \mathsf{u}_{\mathrm{B,C}}(f^{in}_{\mathrm{D}}(T_4), T_4) \tag{21}$$
$$\mathsf{u}_{\mathrm{B,C}}(f^{out}_{\mathrm{D}}(T_9, T_{10}), T_4) \to \mathsf{u}_{\mathrm{C,D}}(f^{in}_{\mathrm{D}}(T_9), T_4, T_9, T_{10}) \tag{22}$$
$$\mathsf{u}_{\mathrm{C,D}}(f^{out}_{\mathrm{D}}(T_{11}, T_8), T_4, T_9, T_{10}) \to f^{out}_{\mathrm{B}}(T_9, T_{10}, T_{11}, T_8) \tag{23}$$
$$f^{in}_{\mathrm{D}}(T_{12}) \to f^{out}_{\mathrm{D}}([\,], T_{12}) \tag{24}$$
$$f^{in}_{\mathrm{D}}(T_{12}) \to \mathsf{u}_{\mathrm{D,G}}(f^{in}_{\mathrm{G}}(T_{12}), T_{12}) \tag{25}$$
$$\mathsf{u}_{\mathrm{D,G}}(f^{out}_{\mathrm{G}}(T_{11}, T_8), T_{12}) \to f^{out}_{\mathrm{D}}(T_{11}, T_8) \tag{26}$$
$$f^{in}_{\mathrm{D}}(T_9) \to \mathsf{u}_{\mathrm{D,F}}(f^{in}_{\mathrm{G}}(T_9), T_9) \tag{27}$$
$$\mathsf{u}_{\mathrm{D,F}}(f^{out}_{\mathrm{G}}(T_{11}, T_8), T_9) \to f^{out}_{\mathrm{D}}(T_{11}, T_8) \tag{28}$$
$$f^{in}_{\mathrm{G}}([T_{14} \,|\, T_{15}]) \to \mathsf{u}_{\mathrm{G,H}}(f^{in}_{\mathrm{D}}(T_{15}), T_{14}, T_{15}) \tag{29}$$
$$\mathsf{u}_{\mathrm{G,H}}(f^{out}_{\mathrm{D}}(T_{16}, T_{13}), T_{14}, T_{15}) \to f^{out}_{\mathrm{G}}([T_{14} \,|\, T_{16}], T_{13}) \tag{30}$$

Its termination is easy to prove by tools like AProVE, *which implies termination of the logic program by Thm. 17. However, this TRS cannot be used for complexity analysis, as $irc_{\mathcal{R}(Gr)}$ is linear whereas the runtime complexity of the original logic program is quadratic. For an analogous reason, complexity analysis of such examples is also not possible by transformations from logic programs to TRSs like [32, 34].*

For complexity analysis, we need a more sophisticated treatment of Split *nodes than for termination analysis. For termination, we only have to approximate the form of the answer substitutions that are computed for the first successor of a* Split *node. This suffices to analyze termination of the evaluations starting in the second successor. However for complexity analysis, we also need to know* how many *answer substitutions are computed for the first successor of a* Split *node, since the evaluation of the second successor is repeated for each such answer substitution. If the first successor of a* Split *node (i.e., a node like* C*) has k answer substitutions, then the evaluation of the second successor (i.e., of* D*) is repeated k times. This is not simulated by the TRS, which replaces backtracking by non-deterministic choice. So after applying rule (21), one has to perform a "first f^{in}_{D}-reduction" to evaluate the f^{in}_{D}-term in the right-hand side to a f^{out}_{D}-term. There exist several possibilities for this reduction (e.g., by using (24), (25), or (27)). So one chooses one such reduction non-deterministically. Afterwards, the remaining rewrite sequence continues with rule (22). However, the TRS does not reflect that in the logic program, one would backtrack afterwards and repeat this remaining rewrite sequence with rule (22), for* every *possible "first f^{in}_{D}-reduction" from $f^{in}_{\mathrm{D}}(\ldots)$ to $f^{out}_{\mathrm{D}}(\ldots)$.*

However, for the star-example of Ex. 2, the first successor G of the only Split node F in the graph of Fig. 6 is deterministic. The reason is that there is at most one answer substitution for any query $\mathsf{app}(t_5, t_7, t_6)$, where t_5 and t_6 are ground terms. In Sect. 6, we will show how to use evaluation graphs in order to analyze determinacy automatically.

Nevertheless, even if all first successors of Split nodes are deterministic, $irc_{\mathcal{R}(Gr)}$ is not necessarily an upper bound of $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}$. This can happen if (i) a Split node s can reach itself via a non-empty path, (ii) its first successor s' reaches a Suc node s'', and (iii) s'' reaches a cycle in the graph.

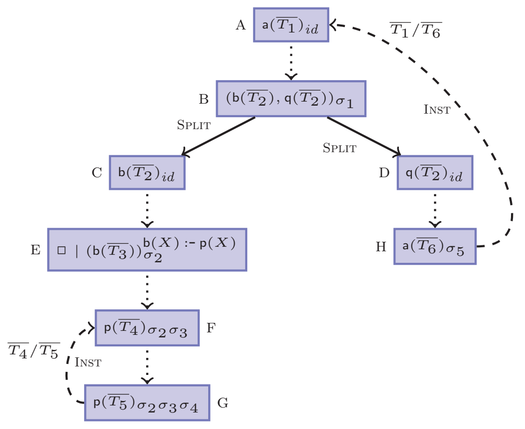


Figure 20: Symbolic Evaluation Graph for Ex. 21

Example 21. *Consider the following program $\mathcal{P}$ and the set of queries $\mathcal{Q}^{\mathsf{a}}_m$ where $m(\mathsf{a}, 1) = in$.*

$$\begin{aligned}
\mathsf{a}(X) &\;\text{:-}\; \mathsf{b}(X), \mathsf{q}(X).\\
\mathsf{b}(X)&.\\
\mathsf{b}(X) &\;\text{:-}\; \mathsf{p}(X).\\
\mathsf{p}(\mathsf{s}(X)) &\;\text{:-}\; \mathsf{p}(X).\\
\mathsf{q}(\mathsf{s}(X)) &\;\text{:-}\; \mathsf{a}(X).
\end{aligned}$$

In the corresponding symbolic evaluation graph in Fig. 20, dotted arrows abbreviate paths of several edges. We have $\sigma_1(\overline{T_1}) = \overline{T_2}$, $\sigma_2(\overline{T_2}) = \overline{T_3}$, $\sigma_3(\overline{T_3}) = \overline{T_4}$, $\sigma_4(\overline{T_4}) = \mathsf{s}(\overline{T_5})$, and $\sigma_5(\overline{T_2}) = \mathsf{s}(\overline{T_6})$. Here, (i) the Split *node* B *reaches itself via a non-empty path, (ii) its first successor* C *reaches a* Suc *node* E*, and (iii)* E *reaches another cycle (from* F *to* G*). The graph has connection paths from* A *to* B*,* C *to* E*,* C *to* F*,* F *to* G*, and* D *to* H*. It results in the following TRS.*

$$f^{in}_{\mathrm{A}}(T_2) \;\to\; \mathsf{u}_{\mathrm{A,B}}(f^{in}_{\mathrm{B}}(T_2), T_2) \tag{31}$$
$$\mathsf{u}_{\mathrm{A,B}}(f^{out}_{\mathrm{B}}, T_2) \;\to\; f^{out}_{\mathrm{A}} \tag{32}$$
$$f^{in}_{\mathrm{B}}(T_2) \;\to\; \mathsf{u}_{\mathrm{B,C}}(f^{in}_{\mathrm{C}}(T_2), T_2) \tag{33}$$
$$\mathsf{u}_{\mathrm{B,C}}(f^{out}_{\mathrm{C}}, T_2) \;\to\; \mathsf{u}_{\mathrm{C,D}}(f^{in}_{\mathrm{D}}(T_2), T_2) \tag{34}$$
$$\mathsf{u}_{\mathrm{C,D}}(f^{out}_{\mathrm{D}}, T_2) \;\to\; f^{out}_{\mathrm{B}} \tag{35}$$
$$f^{in}_{\mathrm{C}}(T_3) \;\to\; f^{out}_{\mathrm{C}} \tag{36}$$
$$f^{in}_{\mathrm{C}}(T_4) \;\to\; \mathsf{u}_{\mathrm{C,F}}(f^{in}_{\mathrm{F}}(T_4), T_4) \tag{37}$$
$$\mathsf{u}_{\mathrm{C,F}}(f^{out}_{\mathrm{F}}, T_4) \;\to\; f^{out}_{\mathrm{C}} \tag{38}$$
$$f^{in}_{\mathrm{F}}(\mathsf{s}(T_5)) \;\to\; \mathsf{u}_{\mathrm{F,G}}(f^{in}_{\mathrm{F}}(T_5), T_5) \tag{39}$$
$$\mathsf{u}_{\mathrm{F,G}}(f^{out}_{\mathrm{F}}, T_5) \;\to\; f^{out}_{\mathrm{F}} \tag{40}$$
$$f^{in}_{\mathrm{D}}(\mathsf{s}(T_6)) \;\to\; \mathsf{u}_{\mathrm{D,H}}(f^{in}_{\mathrm{A}}(T_6), T_6) \tag{41}$$
$$\mathsf{u}_{\mathrm{D,H}}(f^{out}_{\mathrm{A}}, T_6) \;\to\; f^{out}_{\mathrm{D}} \tag{42}$$

For the complexity $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{a}}_m}$ of this program, each call to b *yields both a success (from* C *to* E *in constant time) and a failing further computation (by the cycle from* F *to* G *which takes linear time). Since* b *is called linearly often (by the cycle from* A *to* H*), we obtain a quadratic runtime in total.*

However, the resulting TRS only has linear complexity. Here, the backtracking after the Suc *node* E *is modeled by non-deterministic choice. So to evaluate an f^{in}_{C}-term, one* either *uses rule (36) which corresponds to the path from* C *to* E or *the rules (37), (38) which correspond to the path from*

C *to* F, but not both. *The traversal of the cycle from* A *to* H *can only continue if one evaluates f^{in}_{C} by rule (36), which works in constant time. Only then can the right-hand side of (33) evaluate to the left-hand side of (34).*

Def. 22 captures when $irc_{\mathcal{R}(Gr)}$ is no upper bound of $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}$.

Definition 22 (Multiplicative Split Nodes). *A* Split *node s in a symbolic evaluation graph Gr is called* multiplicative *iff its first successor is not deterministic or if s satisfies the three conditions* (i) – (iii) *above. Let mults(Gr) be the set of all multiplicative* Split *nodes of Gr.*

The only Split node F in the graph of Fig. 6 is indeed non-multiplicative. Its first successor G is deterministic and while F can reach itself via a non-empty path, the only Suc node reachable from its first successor G is J, but J cannot reach a cycle in *Gr* (i.e., (iii) does not hold).

Thm. 23 shows that if the symbolic evaluation graph only contains non-multiplicative Split nodes, our approach can also be used for complexity analysis of logic programs. So the linear complexity of $\mathcal{R}(Gr)$ in our example indeed implies linear complexity of the original program from Ex. 2.

Theorem 23 (Soundness of Complexity Analysis I). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}^{\mathsf{p}}_m$. If Gr has no multiplicative* Split *nodes, then $prc_{\mathcal{P},\mathcal{Q}^{\mathsf{p}}_m}(n) \in \mathcal{O}(irc_{\mathcal{R}(Gr)}(n))$.*

We now extend our approach to also handle examples like Ex. 18 where the evaluation graph *Gr* contains multiplicative Split nodes (i.e., here we have $mults(Gr) = \{\text{B}\}$).

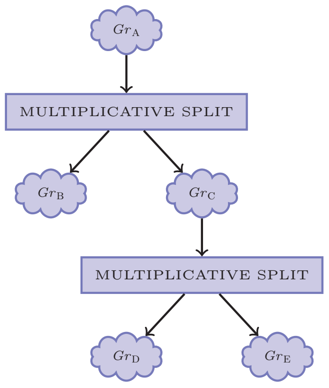
To this end, we generate two *separate* TRSs $\mathcal{R}(Gr_{\text{C}})$ and $\mathcal{R}(Gr_{\text{D}})$ for the subgraphs starting in the two successors C and D of a multiplicative Split node like B in Ex. 18, and multiply their complexity functions $irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}$ and $irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}$. Here, $irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}$ differs from the ordinary complexity function $irc_{\mathcal{R}(Gr)}$ by only counting those rewrite steps that are done with the sub-TRS $\mathcal{R}(Gr_{\text{C}}) \subseteq \mathcal{R}(Gr)$.

In general, for any $\mathcal{R}' \subseteq \mathcal{R}$, the function $irc_{\mathcal{R}',\mathcal{R}}$ maps any $n \in \mathbb{N}$ to the maximal number of $\xrightarrow{\mathsf{i}}_{\mathcal{R}'}$-steps that occur in any sequence of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$-steps starting with a basic term t where $|t| \leq n$. Related notions of "relative" complexity for TRSs were used in, e.g., [4, 20, 31, 41]. Most existing automated complexity provers can also approximate $irc_{\mathcal{R}',\mathcal{R}}$ asymptotically.

The function $irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}$ indeed also yields an upper bound on the number of answer substitutions for C, because the number of answer substitutions cannot be larger than the number of evaluation steps. In our example, both the runtime and the number of answer substitutions for the call $\mathsf{app}(T_5, T_6, \overline{T_4})$ in node C is linear in the size of $\overline{T_4}$'s concretization. Thus, the call $\mathsf{app}(T_{11}, T_8, \overline{T_9})$ in node D, which has linear runtime itself, needs to be repeated a linear number of times. Hence, by multiplying the linear runtime complexities of $irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}$ and $irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}$, we obtain the correct result that the runtime of the original logic program is (at most) quadratic.

So we use the multiplicative Split nodes of a symbolic evaluation graph *Gr* to decompose *Gr* into subgraphs, such that multiplicative Split nodes only occur as the leaves of subgraphs.

As an example, the symbolic evaluation graph on the side is decomposed into the subgraphs $Gr_{\text{A}}, \ldots, Gr_{\text{E}}$ (the subgraphs Gr_{A} and Gr_{C} include the respective multiplicative Split node as a leaf). We now determine the runtime complexities $irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}, \ldots, irc_{\mathcal{R}(Gr_{\text{E}}),\mathcal{R}(Gr)}$ separately and combine them to obtain an upper bound for the runtime of the whole logic program.



As discussed above, the runtime complexity functions resulting from subgraphs of a multiplicative Split node have to be multiplied. In contrast, the runtimes for subgraphs above a multiplicative Split node have to be added. So for the graph on the side, we obtain $irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) + irc_{\mathcal{R}(Gr_{\text{B}}),\mathcal{R}(Gr)}(n) \cdot (irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}(n) + irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}(n) \cdot irc_{\mathcal{R}(Gr_{\text{E}}),\mathcal{R}(Gr)}(n))$ as an approximation for the complexity of the logic program.

To ensure that the symbolic evaluation graph can indeed be decomposed into subgraphs as desired, we have to require that no multiplicative Split node can reach itself again.

Definition 24 (Decomposable Graphs). *A symbolic evaluation graph Gr is called* decomposable *iff there is no non-empty path from a node $s \in mults(Gr)$ to itself.*

The graph in Ex. 18 is decomposable. However, decomposability is a restriction and there are programs in the TPDB whose complexity we cannot analyze, because our graph construction yields a non-decomposable evaluation graph.[9] For instance, the graph in Ex. 21 is not decomposable.

For any node s, the *subgraph at node s* starts in s and stops when reaching multiplicative Split nodes.

Definition 25 (Subgraphs). *Let Gr be a decomposable evaluation graph with nodes V and edges E (i.e., $Gr = (V, E)$) and let $s \in V$. We define the* subgraph of *Gr* at node *s as the minimal graph $Gr_s = (V_s, E_s)$ with $s \in V_s$ that satisfies the following property: whenever $s_1 \in V_s \setminus mults(Gr)$ and $(s_1, s_2) \in E$, then $s_2 \in V_s$ and $(s_1, s_2) \in E_s$.*

Now we decompose the symbolic evaluation graph into the subgraph at the root node and into the subgraphs at all successors of multiplicative Split nodes. So the graph in Ex. 18 is decomposed into Gr_{A}, Gr_{C}, and Gr_{D}, where Gr_{A} contains the 4 nodes from A to B and to ε, Gr_{C} contains all other nodes, and Gr_{D} contains all nodes of Gr_{C} except C.

$\mathcal{R}(Gr_{\text{A}}) = \{(19) - (23)\}$ consists of *ConnectionRules*(π) for the connection path π from A to B and of *SplitRules*(B). For both Gr_{C} and Gr_{D}, we get the same TRS, because C is an instance of D, i.e., $\mathcal{R}(Gr_{\text{C}}) = \mathcal{R}(Gr_{\text{D}}) = \{(24) - (30)\}$.

[9]An extension of our method to examples with non-decomposable evaluation graphs would be an interesting topic for further work. However, even with the restriction to decomposable graphs, our approach is substantially more powerful than all previous techniques for automated complexity analysis of logic programs, cf. the end of this section. In our experiments, there were only 3 examples where other tools could prove an (exponential) upper bound while we failed because of non-decomposability.

For the complexity of the original logic program, we combine the complexities of the sub-TRSs as discussed before. So we multiply the complexities resulting from subgraphs of multiplicative SPLIT nodes, and add all other complexities. The function $cplx_s(n)$ approximates the runtime of the logic program represented by the subgraph of Gr at node s.

DEFINITION 26 (COMPLEXITY FOR SUBGRAPHS). *Let $Gr = (V, E)$ be decomposable. For any $s \in V$ and $n \in \mathbb{N}$, let*

$$cplx_s(n) = \begin{cases} cplx_{Succ_1(s)}(n) \cdot cplx_{Succ_2(s)}(n), \text{ if } s \in mults(Gr) \\ irc_{\mathcal{R}(Gr_s),\mathcal{R}(Gr)}(n) + \\ \quad \Sigma_{s' \in mults(Gr) \cap Gr_s}\ cplx_{s'}(n), \text{ otherwise} \end{cases}$$

So in Ex. 18, we obtain $cplx_{\text{A}}(n) =$

$$\begin{array}{lr} irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) + cplx_{\text{B}}(n) & = \\ irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) + cplx_{\text{C}}(n) \cdot cplx_{\text{D}}(n) & = \\ irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) + irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}(n) \cdot irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}(n) & \end{array}$$

Thm. 27 states that combining the complexities of the TRSs as in Def. 26 indeed yields an upper bound for the complexity of the original logic program.

THEOREM 27 (SOUNDNESS OF COMPLEXITY ANALYSIS II). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$. If Gr is decomposable, then we have $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(cplx_{root(Gr)}(n))$.*

For Ex. 18, tools for complexity analysis of TRSs like TCT and AProVE automatically prove $irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) \in \mathcal{O}(n)$,[10] $irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}(n) \in \mathcal{O}(n)$, $irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}(n) \in \mathcal{O}(n)$. This implies $cplx_{\text{A}}(n) = irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) + irc_{\mathcal{R}(Gr_{\text{C}}),\mathcal{R}(Gr)}(n) \cdot irc_{\mathcal{R}(Gr_{\text{D}}),\mathcal{R}(Gr)}(n) \in \mathcal{O}(n^2)$. Thus, also $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{sublist}}}(n) \in \mathcal{O}(n^2)$.

Note that Thm. 27 subsumes Thm. 23. Every evaluation graph Gr without multiplicative SPLIT nodes is decomposable and here we have $cplx_{root(Gr)}(n) = irc_{\mathcal{R}(Gr)}(n)$.

We also implemented our approach for complexity analysis in our tool AProVE [15]. Existing approaches for direct complexity analysis of logic programs (e.g., [10, 11, 12, 23, 29])[11] are restricted to well-moded logic programs. In contrast, our approach is applicable to a much wider class of logic programs (including non-well-moded and non-definite programs).[12] To compare their power, we evaluated AProVE against the Complexity Analysis System for LOGic (CASLOG) [11] and the Ciao Preprocessor (CiaoPP) [18, 19], which implements the approach of [29]. We ran the three tools on all 477 Prolog programs from the TPDB, again using 2.2 GHz Quad-Opteron 848 Linux machines with a timeout of 60 seconds per program. For CiaoPP we used both the original cost analysis (CiaoPP-o) and CiaoPP's new resource framework which allows to measure different forms of costs (CiaoPP-r). Here, we chose the cost measure "`res_steps`" which approximates the number of resolution steps needed in evaluations.

[10]We even have $irc_{\mathcal{R}(Gr_{\text{A}}),\mathcal{R}(Gr)}(n) \in \mathcal{O}(1)$, i.e., as in Footnote 7, the bounds found by the tools are not always tight.

[11]Some approaches also deduce *lower* complexity bounds for logic programs [12, 23], while we only infer *upper* bounds.

[12]However, our implementation currently does not treat built-in integer arithmetic, while [10, 11, 12, 29] handle linear arithmetic constraints. But our approach could be extended by generating TRSs with built-in integers [14] from the evaluation graphs. This was also done in our approaches for termination analysis of Java via term rewriting [7, 9].

Moreover, we also used CiaoPP to infer the mode and measure information required by CASLOG.

	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \cdot 2^n)$	**bounds**	**RT**
CASLOG	1	21	4	**3**	29	14.8
CiaoPP-o	3	19	4	**3**	29	11.7
CiaoPP-r	3	18	4	**3**	28	12.5
AProVE	**54**	**117**	**37**	0	**208**	**10.6**

In the above table, we used one row for each tool. The first four columns give the number of programs that could be shown to have a constant bound ($\mathcal{O}(1)$), a linear or quadratic polynomial bound ($\mathcal{O}(n)$ or $\mathcal{O}(n^2)$), or an exponential bound ($\mathcal{O}(n \cdot 2^n)$).[13] In column 5 and 6 we give the total number of upper bounds that could be found by the tool and its average runtime on each example. We highlight the best tool for each column using bold font.

The table shows that AProVE can find upper bounds for a much larger subset (42.8%) of the programs than any of the other tools (6.1%). Nevertheless, there are 6 examples where CASLOG or CiaoPP can prove constant (1), linear (1), quadratic (1), or exponential bounds (3), whereas AProVE fails. In summary, the experiments clearly demonstrate that our transformational approach for determining upper bounds advances the state of the art in automated complexity analysis of logic programs significantly.

6. EVALUATION GRAPHS FOR DETERMINACY ANALYSIS

Finally, after having shown how symbolic evaluation graphs can be used for termination and complexity analysis, we consider a third kind of analysis, viz. *determinacy analysis* (cf. the definition of "determinacy" before Ex. 18). Several approaches for determinacy analysis have been developed (e.g., [24, 25, 26, 27, 28, 33]). Moreover, determinacy analysis is also needed for complexity analysis to detect non-deterministic SPLIT nodes in Thm. 23 and 27.

Every successful evaluation corresponds to a path to a SUC node in the evaluation graph. Therefore, this graph is very well suited as a basis for determinacy analysis. A sufficient criterion for determinacy of a state s in the graph is if there is no path starting in s which traverses more than one SUC node. In other words, if s reaches a SUC node s', then there may be no further non-empty path from s' to a SUC node.

THEOREM 28 (SOUNDNESS OF DETERMINACY CRITERION). *Let $\mathcal{P}$ be a logic program and let Gr be a symbolic evaluation graph for $\mathcal{P}$. Let s be a node in Gr such that for all SUC nodes s' reachable from s, there is no non-empty path from s' to a SUC node. Then s is deterministic. Thus, if s is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$ for a $\mathsf{p} \in \Sigma$ and a moding function m, then all queries in $\mathcal{Q}_m^{\mathsf{p}}$ are also deterministic.*

For example, all nodes in the evaluation graph of Fig. 6 satisfy the above determinacy criterion, since there are no non-empty paths from the two SUC nodes E or J to a SUC node again. So the first successor G of the SPLIT node F is deterministic and thus, F is not multiplicative.

In contrast, the node C of the graph in Ex. 18 does not satisfy the determinacy criterion, since it reaches E which has

[13]The back-end of AProVE for complexity analysis of TRSs currently only implements techniques for detecting polynomial bounds. When extending the TRS back-end by other techniques like [5], we could also infer exponential bounds.

a non-empty cycle to itself. Indeed, C is not deterministic and the corresponding SPLIT node B is multiplicative.

Finally, the nodes in the evaluation graph of Ex. 21 are again deterministic, since the only SUC node E has no non-empty path to itself. But since the SPLIT node B satisfies the conditions (i) – (iii), it is nevertheless multiplicative.

Our experiments in Sect. 5 indicate that the criterion of Thm. 28 is strong enough to detect non-multiplicative SPLIT nodes for complexity analysis. But in general, this criterion only represents a first step towards determinacy analysis based on symbolic evaluation graphs and several additional sufficient criteria for determinacy would be possible.

This is also indicated by our experiments when comparing the implementation of our determinacy analysis in AProVE with the determinacy analysis implemented in CiaoPP [27].[14] We again tested both tools on all 477 logic programs from the TPDB. On definite programs, CiaoPP was clearly more powerful (it proved determinacy for 132 out of 300 programs, whereas AProVE only succeeded for 19 programs). But on non-definite programs, AProVE's determinacy analysis is stronger (here, AProVE showed determinacy of 75 out of 177 examples, whereas CiaoPP only succeeded for 61 programs). Altogether, our new determinacy criterion based on evaluation graphs is a substantial addition to existing determinacy analyses, since AProVE succeeded on 58 examples where CiaoPP failed. In other words, by coupling our new technique with existing ones, the power of determinacy analysis can be increased significantly.

7. CONCLUSION

We presented the symbolic evaluation graph and the use of term rewriting as a general methodology for the analysis of logic programs. These graphs represent all evaluations of a (possibly non-definite) logic program in a finite way. Therefore, they can be used as the basis for many different kinds of analyses. In particular, one can translate their paths to rewrite rules and use existing techniques from term rewriting to analyze the termination and complexity of the original logic program. Moreover, one can also perform analyses directly on the evaluation graph (e.g., to examine determinacy).

The current paper does not only give an overview on our previous work on this topic, but it introduces numerous new results. In Sect. 3, we presented a new formulation of the abstract inference rules which is suitable for the subsequent generation of TRSs. Moreover, the theorems of this section (on the connection between concrete and abstract evaluation rules) are new contributions. The approach for termination analysis in Sect. 4 is also substantially different from our earlier approaches, because it directly generates TRSs from evaluation graphs. In particular, this allows us to use the same approach for both termination and complexity analysis. The contributions in Sect. 5 and Sect. 6 (on complexity and determinacy analysis) are completely new.

We implemented all our results in the tool AProVE. Our experiments show that our approaches to termination and complexity analysis are more powerful than previous ones and that our approach to determinacy analysis is a substantial addition to existing ones. See [1] for further details on the experiments and to run AProVE via a web interface.

[14]We did not compare with the determinacy analyzer spdet implemented in SICStus Prolog 4.2.1, since it reports both false positives and false negatives.

Acknowledgements. We thank M. Hermenegildo and P. López-García for their support. Without it, the experimental comparisons with CASLOG and CiaoPP would not have been possible. We also thank N.-W. Lin for agreeing to make the updated version of CASLOG (running under SICStus 4 or Ciao) available on [1].

8. REFERENCES

[1] http://aprove.informatik.rwth-aachen.de/eval/LPGraphs/.

[2] K. R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[3] M. Avanzini, G. Moser, and A. Schnabl. Automated implicit computational complexity analysis. In *Proc. IJCAR '08*, LNAI 5195, pages 132–138, 2008.

[4] M. Avanzini and G. Moser. Dependency pairs and polynomial path orders. In *Proc. RTA '09*, LNCS 5595, pages 48–62, 2009.

[5] M. Avanzini, N. Eguchi, and G. Moser. A path order for rewrite systems that compute exponential time functions. In *Proc. RTA '11*, LIPIcs 10, pages 123–138, 2011.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[7] M. Brockschmidt, C. Otto, and J. Giesl. Modular termination proofs of recursive Java Bytecode programs by term rewriting. In *Proc. RTA '11*, LIPIcs 10, pages 155–170, 2011.

[8] M. Brockschmidt, T. Ströder, C. Otto, and J. Giesl. Automated detection of non-termination and NullPointerExceptions for Java Bytecode. In *Proc. FoVeOOS '11*, LNCS 7421, pages 123–141, 2012.

[9] M. Brockschmidt, R. Musiol, C. Otto, and J. Giesl. Automated termination proofs for Java programs with cyclic data. In *Proc. CAV '12*, LNCS 7358, pages 105–122, 2012.

[10] S. K. Debray, N.-W. Lin, and M. V. Hermenegildo. Task granularity analysis in logic programs. In *Proc. PLDI '90*, pages 174–188. ACM Press, 1990.

[11] S. K. Debray and N.-W. Lin. Cost analysis of logic programs. *ACM Transactions on Programming Languages and Systems*, 15:826–875, 1993.

[12] S. K. Debray, P. López-García, M. V. Hermenegildo, and N.-W. Lin. Lower bound cost estimation for logic programs. In *Proc. ILPS '97*, pages 291–305. MIT Press, 1997.

[13] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1-2):69–116, 1987.

[14] C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In *Proc. RTA '09*, LNCS 5595, pages 32–47, 2009.

[15] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, LNAI 4130, pages 281–286, 2006.

[16] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

[17] J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM*

Transactions on Programming Languages and Systems, 33(2), 2011.

[18] M. V. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Integrated program debugging, verification, and optimization using abstract interpretation (and the Ciao system preprocessor). *Sc. Comp. Prog.*, 58(1-2):115–140, 2005.

[19] M. V. Hermenegildo, F. Bueno, M. Carro, P. López-García, E. Mera, J. F. Morales, and G. Puebla. An overview of Ciao and its design philosophy. *Theory and Practice of Logic Programming*, 12(1-2):219–252, 2012.

[20] N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, LNAI 5195, pages 364–379, 2008.

[21] J. M. Howe and A. King. Efficient groundness analysis in Prolog. *Th. Pract. Log. Prog.*, 3(1):95–124, 2003.

[22] ISO/IEC 13211-1. *Information technology - Programming languages - Prolog*. 1995.

[23] A. King, K. Shen, and F. Benoy. Lower-bound time-complexity analysis of logic programs. In *Proc. ILPS '97*, pages 261–285. MIT Press, 1997.

[24] A. King, L. Lu, and S. Genaim. Detecting determinacy in Prolog programs. In *Proc. ICLP '06*, LNCS 4079, pages 132–147, 2006.

[25] J. Kriener and A. King. RedAlert: Determinacy inference for Prolog. *In Proc. ICLP '11, Theory and Practice of Logic Programming*, 11(4-5):537–553, 2011.

[26] J. Kriener and A. King. Mutual exclusion by interpolation. In *Proc. FLOPS '12*, LNCS 7294, pages 182–196, 2012.

[27] P. López-García, F. Bueno, and M. V. Hermenegildo. Automatic inference of determinacy and mutual exclusion for logic programs using mode and type analyses. *New Generation Comp.*, 28(2):177–206, 2010.

[28] T. Mogensen. A semantics-based determinacy analysis for Prolog with cut. In *Proc. Ershov Memorial Conference '96*, LNCS 1181, pages 374–385, 1996.

[29] J. A. Navas, E. Mera, P. López-García, and M. V. Hermenegildo. User-definable resource bounds analysis for logic programs. In *Proc. ICLP '07*, LNCS 4670, pages 348–363, 2007.

[30] M. T. Nguyen, J. Giesl, and P. Schneider-Kamp. Termination analysis of logic programs based on dependency graphs. In *Proc. LOPSTR '07*, LNCS 4915, pages 8–22, 2008.

[31] L. Noschinski, F. Emmes, and J. Giesl. The dependency pair framework for automated complexity analysis of term rewrite systems. In *Proc. CADE '11*, LNAI 6803, pages 422–438, 2011.

[32] E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing*, 12(1-2):73–116, 2001.

[33] D. Sahlin. Determinacy analysis for full Prolog. In *Proc. PEPM '91*, pages 23–30. ACM Press, 1991.

[34] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic*, 11(1), 2009.

[35] P. Schneider-Kamp, J. Giesl, and M. T. Nguyen. The dependency triple framework for termination of logic programs. In *Proc. LOPSTR '09*, LNCS 6037, pages 37–51, 2010.

[36] P. Schneider-Kamp, J. Giesl, T. Ströder, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs with cut. *In Proc. ICLP '10, Theory and Practice of Logic Programming*, 10(4-6):365–381, 2010.

[37] M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In *Proc. ILPS '95*, pages 465–479. MIT Press, 1995.

[38] T. Ströder. Towards termination analysis of real Prolog programs. Diploma Thesis, RWTH Aachen, 2010. Available from [1].

[39] T. Ströder, P. Schneider-Kamp, and J. Giesl. Dependency triples for improving termination analysis of logic programs with cut. In *Proc. LOPSTR '10*, LNCS 6564, pages 184–199, 2011.

[40] T. Ströder, F. Emmes, P. Schneider-Kamp, J. Giesl, and C. Fuhs. A linear operational semantics for termination and complexity analysis of ISO Prolog. In *Proc. LOPSTR '11*, 2012. To appear. Available from [1].

[41] H. Zankl and M. Korp. Modular complexity analysis via relative complexity. In *Proc. RTA '10*, LIPIcs 6, pages 385–400, 2010.

[42] H. Zantema. Termination. In Terese, editor, *Term Rewriting Systems*, pages 181–259. Cambridge University Press, 2003.

APPENDIX

A. PROOFS

Before proving Thm. 5, we introduce a notation for evaluations of length k where we make the applied unifiers explicit. This generalizes the notion of *answer substitutions* from Def. 3 to evaluations ending in arbitrary states (i.e., in states where the first goal may also be different from $\square_{\sigma\theta}$)

DEFINITION 29 ($\vdash^k_\theta$). *Let S be a state with only one single goal. We say that there is an evaluation of length k from S to another state S' using the unifier θ (denoted $S \vdash^k_\theta S'$) if*

- *S' results from S by applying rules from Def. 1 k-times,*
- *S is of the form Q_σ for some query Q (which may additionally be labeled by a clause c), and*
- *S' is of the form $(Q'_{\sigma\theta} \mid S_{\mathit{suffix}})$ for some query Q' (which may additionally be labeled by a clause c') and a (possibly empty) state S_{suffix}.*

Now we prove Thm. 5. For all rules from Def. 4 except EVAL, the proof is straightforward. This is also true for the backtracking case of the EVAL rule (i.e., for the second successor in the EVAL rule). The intuitive reason why the EVAL rule is also sound in the case where the unification succeeds by an mgu σ is that σ only has fresh variables in its range. Thus, if $mgu(t\gamma, h)$ also exists for some concretization γ, then the composition of γ and $mgu(t\gamma, h)$ is an instance of σ. Moreover, we can also apply $\sigma|_{\mathcal{G}}$ to all remaining goals in the state, as the terms represented by variables from $\mathcal{G}$ do not contain variables. Hence, $\sigma|_{\mathcal{G}}$ does not correspond to an instantiation of concrete variables, but only to a case analysis on the shape of the ground terms represented by abstract variables in $\mathcal{G}$. For example, replacing a variable $T \in \mathcal{G}$ by the term $\mathsf{s}(T')$ using the unifier σ corresponds to the situation that the unification only succeeds if T represented a term of the form $\mathsf{s}(t')$ where t' is some ground term.

THEOREM 5 (SOUNDNESS OF ABSTRACT RULES). *Let $(S; KB)$ be an abstract state with a concretization $S\gamma \in \mathcal{CON}(S; KB)$, and let S_{next} be the successor of $S\gamma$ according to the operational semantics in Def. 1. Then the abstract state $(S; KB)$ has a successor $(S'; KB')$ according to an inference rule from Def. 4 such that $S_{next} \in \mathcal{CON}(S'; KB')$.*

PROOF. For the SUC rule, we have $S = (\square_{\theta'} \mid S_{\mathit{suffix}})$, $S\gamma = (\square_\theta \mid S_{\mathit{suffix}}\gamma)$, and $S_{next} = S_{\mathit{suffix}}\gamma$ for some substitutions θ and θ'. The successor of $(S; KB)$ is $(S_{\mathit{suffix}}; KB)$. So obviously we have $S_{next} \in \mathcal{CON}(S_{\mathit{suffix}}; KB)$.

For the first CUT rule, we have $S = ((!_m, Q)_{\theta'} \mid S_1 \mid ?_m \mid S_2)$, $S\gamma = ((!_m, Q\gamma)_\theta \mid S_1\gamma \mid ?_m \mid S_2\gamma)$, and $S_{next} = ((Q\gamma)_\theta \mid ?_m \mid S_2\gamma)$ for some substitutions θ and θ', where S_1 does not contain $?_m$. The successor of $(S; KB)$ is $(Q_{\theta'} \mid ?_m \mid S_2; KB)$. So obviously we have $S_{next} \in \mathcal{CON}(Q_{\theta'} \mid ?_m \mid S_2; KB)$. For the second CUT rule, we have $S = ((!_m, Q)_{\theta'} \mid S_{\mathit{suffix}})$, $S\gamma = ((!_m, Q\gamma)_\theta \mid S_{\mathit{suffix}}\gamma)$, and $S_{next} = (Q\gamma)_\theta$ for some substitutions θ and θ', where S_{suffix} does not contain $?_m$. The successor of $(S; KB)$ is $(Q_{\theta'}; KB)$. So obviously we have $S_{next} \in \mathcal{CON}(Q_{\theta'}; KB)$.

For the CASE rule, we have $S = ((t, Q)_{\theta'} \mid S_{\mathit{suffix}})$, $S\gamma = ((t\gamma, Q\gamma)_\theta \mid S_{\mathit{suffix}}\gamma)$, and $S_{next} = ((t\gamma, Q\gamma)^{c_1[!/!_m]}_\theta \mid \cdots \mid (t\gamma, Q\gamma)^{c_a[!/!_m]}_\theta \mid S_{\mathit{suffix}}\gamma)$ for some substitutions θ and θ', where t is neither a cut nor a variable, m is fresh, and $\mathit{Slice}(\mathcal{P}, t) = (c_1, \ldots, c_a)$. The successor of $(S; KB)$ is $((t, Q)^{c_1[!/!_m]}_{\theta'} \mid \cdots \mid (t, Q)^{c_a[!/!_m]}_{\theta'} \mid S_{\mathit{suffix}}; KB)$. So obviously we have $S_{next} \in \mathcal{CON}((t, Q)^{c_1[!/!_m]}_{\theta'} \mid \cdots \mid (t, Q)^{c_a[!/!_m]}_{\theta'} \mid S_{\mathit{suffix}}; KB)$.

For the FAIL rule, we have $S = (?_m \mid S_{\mathit{suffix}})$, $S\gamma = (?_m \mid S_{\mathit{suffix}}\gamma)$, and $S_{next} = S_{\mathit{suffix}}\gamma$. The successor of $(S; KB)$ is $(S_{\mathit{suffix}}; KB)$. So obviously $S_{next} \in \mathcal{CON}(S_{\mathit{suffix}}; KB)$.

For the BACKTRACK rule, we have $S = ((t, Q)^{h\,:\!-\,B}_{\theta'} \mid S_{\mathit{suffix}})$, $S\gamma = ((t\gamma, Q\gamma)^{h\,:\!-\,B}_\theta \mid S_{\mathit{suffix}}\gamma)$ for some substitutions θ and θ', and $S_{next} = S_{\mathit{suffix}}\gamma$, because we know that $t\gamma \nsim h$. The successor of $(S; KB)$ is $(S_{\mathit{suffix}}; KB)$. So obviously we have $S_{next} \in \mathcal{CON}(S_{\mathit{suffix}}; KB)$.

For the EVAL rule, we have $S = ((t, Q)^{h\,:\!-\,B}_{\theta'} \mid S_{\mathit{suffix}})$, $KB = (\mathcal{G}, \mathcal{U})$, and $S\gamma = ((t\gamma, Q\gamma)^{h\,:\!-\,B}_\theta \mid S_{\mathit{suffix}}\gamma)$. There are two cases depending on whether $t\gamma$ and h unify.

First, if $t\gamma$ does not unify with h, then $S\gamma$ has to be evaluated using the BACKTRACK rule and we obtain $S_{next} = S_{\mathit{suffix}}\gamma$. As γ is a concretization and h does not contain abstract variables, we have $h\gamma = h$ and, thus, $t\gamma \nsim h\gamma$. Hence, γ is also a concretization w.r.t. the knowledge base $(\mathcal{G}, \mathcal{U} \cup \{(t, h)\})$. So we have $S_{next} \in \mathcal{CON}(S_{\mathit{suffix}}; (\mathcal{G}, \mathcal{U} \cup \{(t, h)\}))$ for the second successor $(S_{\mathit{suffix}}; (\mathcal{G}, \mathcal{U} \cup \{(t, h)\}))$ of $(S; KB)$.

Second, let $mgu(t\gamma, h) = \theta''$. W.l.o.g., we assume $\mathcal{V}(\mathit{Range}(\theta'')) \subseteq \mathcal{N}$. Thus, $S\gamma$ has to be evaluated using the EVAL rule and we have $S_{next} = ((B\theta'', Q\gamma\theta'')_{\theta\theta''} \mid S_{\mathit{suffix}}\gamma)$. From $h\gamma = h$ we know that $mgu(t\gamma, h\gamma) = \theta''$, too. Thus, there is a substitution σ with $mgu(t, h) = \sigma$, where $\mathcal{V}(\sigma(X))$ only contains fresh abstract variables for all $X \in \mathcal{V}$. Moreover, $\gamma\theta''$ is a unifier of t and h and, thus, there is a substitution ξ with $\sigma\xi = \gamma\theta''$. The first successor of $(S; KB)$ is $((B\sigma, Q\sigma)_{\theta'\sigma} \mid S'_{\mathit{suffix}}; (\mathcal{G}', \mathcal{U}\sigma|_{\mathcal{G}}))$ with $\mathcal{G}' = \mathcal{A}(\mathit{Range}(\sigma|_{\mathcal{G}}))$ and S'_{suffix} results from S_{suffix} by applying the substitution $\sigma|_{\mathcal{G}}$ to all its goals and composing $\sigma|_{\mathcal{G}}$ with the substitution labels of its goals. We are, thus, left to show that $S_{next} \in \mathcal{CON}((B\sigma, Q\sigma)_{\theta'\sigma} \mid S'_{\mathit{suffix}}; (\mathcal{G}', \mathcal{U}\sigma|_{\mathcal{G}}))$, i.e., that there is a concretization γ' w.r.t. $(\mathcal{G}', \mathcal{U}\sigma|_{\mathcal{G}})$ such that $B\theta'' = B\sigma\gamma'$, $Q\gamma\theta'' = Q\sigma\gamma'$, and $S_{\mathit{suffix}}\gamma = S'_{\mathit{suffix}}\gamma'$.

We define $\gamma'(T) = \xi(T)$ for $T \in \mathcal{A}(\mathit{Range}(\sigma))$ and $\gamma'(T) = \gamma(T)$ for $T \in \mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\sigma))$. Obviously, we have $\mathit{Dom}(\gamma') = \mathcal{A}$.

We continue by showing that γ' is a concretization w.r.t. $(\mathcal{G}', \mathcal{U}\sigma|_{\mathcal{G}})$, i.e., that $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$, $\mathcal{V}(\mathit{Range}(\gamma'|_{\mathcal{G}'})) = \varnothing$, and $\bigwedge_{(s',t') \in \mathcal{U}\sigma|_{\mathcal{G}}} s'\gamma' \nsim t'\gamma'$.

We first show that $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$. In other words, for all $T \in \mathcal{A}$, we show that $\mathcal{V}(T\gamma') \subseteq \mathcal{N}$. To this end, we perform a case analysis w.r.t. $\mathcal{A} = \mathcal{A}(\mathit{Range}(\sigma)) \uplus (\mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\sigma)))$. For $T \in \mathcal{A}(\mathit{Range}(\sigma))$ we have $\mathcal{V}(T\gamma') = \mathcal{V}(T\xi)$, by the definition of γ'. Assume there is a $T' \in \mathcal{A} \cap \mathcal{V}(T\xi)$. As $T \in \mathcal{A}(\mathit{Range}(\sigma))$, there is an $X \in \mathit{Dom}(\sigma)$ with $T \in \mathcal{V}(X\sigma)$ and, thus, $T' \in \mathcal{V}(X\sigma\xi) = \mathcal{V}(X\gamma\theta'')$. However, we have $\mathcal{V}(X\gamma) \subseteq \mathcal{N}$ and $\mathcal{V}(\mathit{Range}(\theta'')) \subseteq \mathcal{N}$, which leads to a contradiction. Thus, we must have $\mathcal{V}(T\gamma') = \mathcal{V}(T\xi) \subseteq \mathcal{N}$. For $T \in \mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\sigma))$ we have $\mathcal{V}(T\gamma') = \mathcal{V}(T\gamma)$ by the definition of γ' and $\mathcal{V}(T\gamma) \subseteq \mathcal{N}$, since γ is a concretization.

To show that $\mathcal{V}(\mathit{Range}(\gamma'|_{\mathcal{G}'})) = \varnothing$, let $T \in \mathcal{G}' = \mathcal{A}(\mathit{Range}(\sigma|_{\mathcal{G}}))$. Our goal is to show that $\mathcal{V}(T\gamma') = \varnothing$. Recall that $\mathcal{V}(T\gamma') = \mathcal{V}(T\xi)$ for such T. Moreover, there must be a $T' \in \mathit{Dom}(\sigma|_{\mathcal{G}})$ with $T \in \mathcal{V}(T'\sigma)$. Hence, it suffices to show that $\mathcal{V}(T\gamma') = \mathcal{V}(T\xi) \subseteq \mathcal{V}(T'\sigma\xi) = \varnothing$. By the definition of ξ, we have $\mathcal{V}(T'\sigma\xi) = \mathcal{V}(T'\gamma\theta'')$ and $\mathcal{V}(T'\gamma\theta'') = \varnothing$ holds since $T' \in \mathcal{G}$ and γ is a concretization w.r.t. $(\mathcal{G}, \mathcal{U})$.

Finally, we show that $\bigwedge_{(s',t') \in \mathcal{U}\sigma|_{\mathcal{G}}} s'\gamma' \nsim t'\gamma'$. Note that

$\bigwedge_{(s',t') \in \mathcal{U}\sigma|_{\mathcal{G}}} s'\gamma' \nsim t'\gamma'$ holds iff $\bigwedge_{(s',t') \in \mathcal{U}} s'\sigma|_{\mathcal{G}}\gamma' \nsim t'\sigma|_{\mathcal{G}}\gamma'$ holds, which in turn holds iff $\bigwedge_{(s',t') \in \mathcal{U}} s'\gamma \nsim t'\gamma$ holds. (The truth of the last statement follows from the fact that γ is a concretization w.r.t. $(\mathcal{G},\mathcal{U})$.) The reason for the last equivalence above is that $T\sigma|_{\mathcal{G}}\gamma' = T\gamma$ for all $T \in \mathcal{V}(\mathcal{U})$. For non-abstract variables $T \in \mathcal{N}$, this is clear since they are not contained in $\mathcal{G}$, and thus they are not modified by $\sigma|_{\mathcal{G}}$, γ', or γ. To see why $T\sigma|_{\mathcal{G}}\gamma' = T\gamma$ holds for all $T \in \mathcal{A}(\mathcal{U})$, note that $T \notin \mathcal{V}(\mathit{Range}(\sigma))$ as $\mathcal{V}(\sigma(X))$ only contains fresh abstract variables for all previously occurring $X \in \mathcal{V}$. Now consider the partition $\mathcal{A}(\mathcal{U}) = (\mathcal{A}(\mathcal{U}) \setminus \mathcal{G}) \uplus (\mathcal{A}(\mathcal{U}) \cap \mathcal{G})$. If $T \in \mathcal{A}(\mathcal{U}) \setminus \mathcal{G}$, we have $T\gamma = T\gamma'$ by the definition of γ' as $T \notin \mathcal{V}(\mathit{Range}(\sigma))$, and $T\gamma' = T\sigma|_{\mathcal{G}}\gamma'$ since $T \notin \mathit{Dom}(\sigma|_{\mathcal{G}})$. If $T \in \mathcal{A}(\mathcal{U}) \cap \mathcal{G}$, we have $T\gamma = T\gamma\theta''$, since $T \in \mathcal{G}$. Moreover, $T\gamma\theta'' = T\sigma\xi$ by the definition of ξ, and $T\sigma\xi = T\sigma|_{\mathcal{G}}\xi$ since $T \in \mathcal{G}$. As $\mathcal{V}(\mathit{Range}(\sigma|_{\mathcal{G}})) \subseteq \mathcal{A}$, by the definition of γ' we obtain $T\sigma|_{\mathcal{G}}\xi = T\sigma|_{\mathcal{G}}\gamma'$.

Now it remains to show that $B\theta'' = B\sigma\gamma'$, $Q\gamma\theta'' = Q\sigma\gamma'$, and $S_{\mathit{suffix}}\gamma = S'_{\mathit{suffix}}\gamma'$.

Obviously, we choose the same substitution labels in the concretizations $S_{\mathit{suffix}}\gamma$ and $S'_{\mathit{suffix}}\gamma'$. So for the remaining proof, we disregard substitution labels and have $S'_{\mathit{suffix}} = S_{\mathit{suffix}}\sigma|_{\mathcal{G}}$. Thus, we have to prove $S_{\mathit{suffix}}\gamma = S_{\mathit{suffix}}\sigma|_{\mathcal{G}}\gamma'$. This follows from the fact that $T\gamma = T\sigma|_{\mathcal{G}}\gamma'$ holds for all $T \in \mathcal{A}(S)$, which can be proved analogously to the analysis for $\mathcal{A}(\mathcal{U})$ above.

For B, we obtain $B\theta'' = B\gamma\theta''$ as $\mathcal{V}(B) \cap \mathcal{A} = \varnothing$. By the definition of ξ, we have $B\gamma\theta'' = B\sigma\xi$. Since σ renames all occurring variables to fresh abstract variables, we have $\mathcal{V}(B\sigma) \subseteq \mathcal{A}(\mathit{Range}(\sigma))$ and thus, $B\sigma\xi = B\sigma\gamma'$ by the definition of γ'.

Analogously, $Q\gamma\theta'' = Q\sigma\xi$ by the definition of ξ. Since $\mathcal{V}(Q\sigma) \subseteq \mathcal{A}(\mathit{Range}(\sigma))$, too, we have $Q\sigma\xi = Q\sigma\gamma'$ by the definition of γ'. $\square$

Next we prove the soundness of the Inst rule. The reason for its soundness is that the matcher μ corresponds to a restriction of the shapes of the represented terms. Thus, the first state in the Inst rule represents a subset of the concrete states represented by the second state in the rule.

Theorem 8 (Soundness of Inst). *Let $(S; KB)$ be an abstract state, let $(S'; KB')$ be its successor according to the* Inst *rule, and let μ be the substitution associated to $(S; KB)$. If $S\gamma \in \mathcal{CON}(S; KB)$, then for $\gamma' = \mu\gamma$ we have $S'\gamma' = S\gamma \in \mathcal{CON}(S'; KB')$.*

Proof. Let $S\gamma \in \mathcal{CON}(S; KB)$ and $KB = (\mathcal{G},\mathcal{U})$. We first show that $\gamma' = \mu\gamma$ is a concretization w.r.t. $KB' = (\mathcal{G}',\mathcal{U}')$, i.e., that $\mathit{Dom}(\gamma') = \mathcal{A}$, $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$, $\mathcal{V}(\mathit{Range}(\gamma'|_{\mathcal{G}'})) = \varnothing$, and $\bigwedge_{(s,t)\in\mathcal{U}'} s\gamma' \nsim t\gamma'$.

As $\mathit{Dom}(\mu) \subseteq \mathcal{A}$ and $\mathit{Dom}(\gamma) = \mathcal{A}$, we obviously have $\mathit{Dom}(\gamma') = \mathcal{A}$.

From $\mathit{Dom}(\gamma) = \mathcal{A}$ and $\mathcal{V}(\mathit{Range}(\gamma)) \subseteq \mathcal{N}$, we also obtain $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$.

Let $T' \in \mathcal{G}'$. We have to show that $\mathcal{V}(T'\gamma') = \mathcal{V}(T'\mu\gamma) = \varnothing$. By the definition of $\mathcal{G}'$, we have $\mathcal{V}(T'\mu) \subseteq \mathcal{G}$ and thus, $\mathcal{V}(T'\mu\gamma) = \varnothing$ as γ is a concretization w.r.t. $(\mathcal{G},\mathcal{U})$.

Finally, from $\bigwedge_{(s,t)\in\mathcal{U}} s\gamma \nsim t\gamma$ and $\mathcal{U}'\mu \subseteq \mathcal{U}$, we know that $\bigwedge_{(s,t)\in\mathcal{U}'\mu} s\gamma \nsim t\gamma$. This implies $\bigwedge_{(s,t)\in\mathcal{U}'} s\mu\gamma \nsim t\mu\gamma$ and thus, $\bigwedge_{(s,t)\in\mathcal{U}'} s\gamma' \nsim t\gamma'$.

We are left to show that $S'\gamma' = S\gamma$ (where we choose the same substitution labels in $S'\gamma'$ and $S\gamma$). By Def. 7 we know that $S = Q_\theta$ and $S' = Q'_{\theta'}$ or $S = Q^c_\theta$ and $S' = Q'^c_{\theta'}$ for some non-empty queries Q and Q', some substitutions θ and θ', and some clause c. Thus, to show that $S'\gamma' = S\gamma$, we can disregard substitution and clause labels and are left to show that $Q\gamma = Q'\gamma' = Q'\mu\gamma$. This follows immediately from Def. 7 as we have $Q = Q'\mu$. $\square$

The key to the soundness of the Split rule is the substitution δ which over-approximates all possible answer substitutions for the first successor of a Split node. This over-approximation is realized by replacing every variable by a fresh one. In this way, any possible instantiation by an answer substitution is represented. The only remaining restriction is that the fresh variables are still constrained by the updated knowledge base.

Theorem 10 (Soundness of Split). *Let $((t,Q)_\theta;KB)$ be an abstract state and let $(t_{id}; KB)$ and $((Q\delta)_\delta; KB')$ be its successors according to the* Split *rule. Let $(t,Q)\gamma \in \mathcal{CON}((t,Q)_\theta; KB)$ and let θ' be an answer substitution of $(t\gamma)_{id}$. Then there is a concretization γ' w.r.t. KB' such that $Q\gamma\theta' = Q\delta\gamma'$ and $t\gamma\theta' = t\delta\gamma'$. In particular, we have $Q\gamma\theta' \in \mathcal{CON}((Q\delta)_\delta; KB')$.*

Proof. W.l.o.g., we demand $\mathcal{V}(\mathit{Range}(\theta')) \subseteq \mathcal{N}$. Let $KB = (\mathcal{G},\mathcal{U})$, and V contain all abstract variables occurring in $((t,Q)_\theta; KB)$. Then $\delta|_{V\setminus\mathcal{G}}$ must be injective as the variables in its range are fresh. So there exists a substitution δ^{-1} with $\delta^{-1}(T') = T$ if there is a $T \in V \setminus \mathcal{G}$ with $\delta(T) = T'$ and $\delta^{-1}(T') = T'$ for all other variables T'. We define $T\gamma' = T\gamma\theta'$ for $T \in \mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$ and $T\gamma' = T\delta^{-1}\gamma\theta'$ for $T \in \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$. We have $KB' = (\mathcal{G}',\mathcal{U}') = (\mathcal{G} \cup \mathit{NextG}(t,\mathcal{G})\delta, \mathcal{U}\delta)$. We are left to show that γ' is a concretization w.r.t. KB', that $Q\gamma\theta' = Q\delta\gamma'$, and that $t\gamma\theta' = t\delta\gamma'$.

We first show that γ' is a concretization w.r.t. KB', i.e., that $\mathit{Dom}(\gamma') = \mathcal{A}$, $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$, $\mathcal{V}(\mathit{Range}(\gamma'|_{\mathcal{G}'})) = \varnothing$, and $\bigwedge_{(s,t)\in\mathcal{U}'} s\gamma' \nsim t\gamma'$.

Obviously, we have defined γ' such that $\mathit{Dom}(\gamma') = \mathcal{A}$.

To show $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$, we first note that we have $\mathit{Dom}(\gamma) = \mathcal{A}$, $\mathcal{V}(\mathit{Range}(\gamma)) \subseteq \mathcal{N}$, and $\mathcal{V}(\mathit{Range}(\theta')) \subseteq \mathcal{N}$. Thus, we directly obtain $\mathcal{V}(\mathit{Range}(\gamma')) \subseteq \mathcal{N}$.

To prove $\mathcal{V}(\mathit{Range}(\gamma'|_{\mathcal{G}'})) = \varnothing$, we perform a case analysis w.r.t. the partition $\mathcal{G}' = ((\mathcal{G} \cup \mathit{NextG}(t,\mathcal{G})\delta) \setminus \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))) \uplus ((\mathcal{G} \cup \mathit{NextG}(t,\mathcal{G})\delta) \cap \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}})))$.

Let $T \in (\mathcal{G} \cup \mathit{NextG}(t,\mathcal{G})\delta) \setminus \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$. We have to show that $\mathcal{V}(T\gamma') = \varnothing$. Note that by the definition of γ', we have $T\gamma' = T\gamma\theta'$, i.e., we have to show $\mathcal{V}(T\gamma\theta') = \varnothing$. If $T \in \mathcal{G}$, then this is obvious as γ is a concretization w.r.t. $(\mathcal{G},\mathcal{U})$. Note that if $T \notin \mathcal{G}$, then we also cannot have $T \in \mathit{NextG}(t,\mathcal{G})\delta$. The reason is that then there would have to be a $T' \in \mathit{NextG}(t,\mathcal{G})$ with $T = T'\delta$. But as $T \notin \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$, this means that $T' \notin V \setminus \mathcal{G}$. Since $\mathcal{A}(\mathit{NextG}(t,\mathcal{G})) \subseteq \mathcal{A}(t) \subseteq V$, this means $T' \in \mathcal{G}$. But then we have $T = T'\delta = T'$ (i.e., $T \in \mathcal{G}$).

Now let $T \in (\mathcal{G} \cup \mathit{NextG}(t,\mathcal{G})\delta) \cap \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$. Again, we have to show that $\mathcal{V}(T\gamma') = \varnothing$. By the definition of γ', we have $T\gamma' = T\delta^{-1}\gamma\theta'$, i.e., we have to show $\mathcal{V}(T\delta^{-1}\gamma\theta') = \varnothing$. By the definition of δ^{-1}, we know that there is a $T' \in V \setminus \mathcal{G}$ with $T'\delta = T$ and $T\delta^{-1} = T'$. Thus, we now have to show $\mathcal{V}(T'\gamma\theta') = \varnothing$. Note that since δ replaces T' by a fresh variable T, we have $T \notin \mathcal{G}$. Thus, $T \in \mathit{NextG}(t,\mathcal{G})\delta$, i.e., $T' \in \mathit{NextG}(t,\mathcal{G})$. Let $t = \mathsf{p}(t_1,\ldots,t_k)$. As γ is a concretization w.r.t. $(\mathcal{G},\mathcal{U})$, we have $\mathcal{V}(t_i\gamma) = \varnothing$ for all i

where $\mathcal{V}(t_i) \subseteq \mathcal{G}$. As θ' is an answer substitution of $t\gamma$, $\mathcal{V}(t_j\gamma\theta') = \varnothing$ for all $j \in \mathit{Ground}_{\mathcal{P}}(\mathsf{p},\ \{i \mid \mathcal{V}(t_i) \subseteq \mathcal{G}\})$. Now $T' \in \mathit{NextG}(t, \mathcal{G})$ means that $T' \in \mathcal{V}(t_j)$ for some $j \in \mathit{Ground}_{\mathcal{P}}(\mathsf{p},\ \{i \mid \mathcal{V}(t_i) \subseteq \mathcal{G}\})$. Hence, $\mathcal{V}(T'\gamma\theta') = \varnothing$.

Finally, recall that $\mathcal{U}' = \mathcal{U}\delta$. Thus, $\bigwedge_{(s,t)\in\mathcal{U}'} s\gamma' \not\approx t\gamma'$ iff $\bigwedge_{(s,t)\in\mathcal{U}\delta} s\gamma' \not\approx t\gamma'$ iff $\bigwedge_{(s,t)\in\mathcal{U}} s\delta\gamma' \not\approx t\delta\gamma'$. We now show that we have $s\delta\gamma' = s(\gamma\theta')|_{\mathcal{A}}$ and $t\delta\gamma' = t(\gamma\theta')|_{\mathcal{A}}$. Note that $(\gamma\theta')|_{\mathcal{A}} = \gamma\theta'|_{\mathit{Range}(\gamma)}$. Thus, $\bigwedge_{(s,t)\in\mathcal{U}} s\delta\gamma' \not\approx t\delta\gamma'$ follows from $\bigwedge_{(s,t)\in\mathcal{U}} s\gamma \not\approx t\gamma$ (which holds, since γ is a concretization w.r.t. $(\mathcal{G},\mathcal{U})$).

To see why $s\delta\gamma' = s(\gamma\theta')|_{\mathcal{A}}$ and $t\delta\gamma' = t(\gamma\theta')|_{\mathcal{A}}$ holds, note that the abstract variables in s and t are in V. So it suffices to show that for all $T \in V$, we have $T\delta\gamma' = T\gamma\theta'$. Consider the partition $V = (V \setminus \mathit{Dom}(\delta)) \uplus (V \cap \mathit{Dom}(\delta))$. For $T \in V \setminus \mathit{Dom}(\delta)$, we know that $T \in \mathcal{G}$, because δ replaces all variables in $V \setminus \mathcal{G}$ by fresh abstract variables. In particular, T is not fresh and, thus, cannot occur in the range of δ. So we have $T\delta\gamma' = T\gamma' = T\gamma\theta'$ by the definition of γ'. For $T \in V \cap \mathit{Dom}(\delta)$, we know that $T\delta \in \mathcal{A}(\mathit{Range}(\delta|_{V\setminus\mathcal{G}}))$ and $T \notin \mathcal{G}$. Hence, we obtain $T\delta\gamma' = T\delta\delta^{-1}\gamma\theta'$ by the definition of γ'. Thus, $T\delta\delta^{-1}\gamma\theta' = T\gamma\theta'$ by the definition of δ^{-1}.

We are left to show that $Q\gamma\theta' = Q\delta\gamma'$ and $t\gamma\theta' = t\delta\gamma'$ holds. Since $\mathcal{V}(Q) \cup \mathcal{V}(t) \subseteq V$, we obtain $Q\gamma\theta' = Q\delta\gamma'$ and $t\gamma\theta' = t\delta\gamma'$ by an analogous reasoning as above. $\square$

To prove Thm. 16, we need a lemma on the connection between the substitution labels in the abstract and in the concrete case.

The substitution σ_π which describes the difference between the substitution label of the first and the last node of a connection path π indeed approximates the substitutions applied during a concrete evaluation following such a path. In other words, σ_π in the abstract case plays the same role as the unifiers θ in concrete evaluations of the form $S \vdash_\theta^k S'$ in Def. 3.

More precisely, the situation for a connection path π is that we have a concretization γ for the start node $s = (S; \mathit{KB})$ of π yielding a concrete state $S\gamma$, a concretization γ' for the end node $s' = (S'; \mathit{KB}')$ of π yielding a concrete state $S'\gamma'$, a substitution σ_π which matches the substitution label of the goal of s to the substitution label of the first goal of s', and a substitution θ obtained during the concrete evaluation from $S\gamma$ to $S'\gamma'$. Then we must have $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta)|_{\mathcal{V}(s)}$ as illustrated in the following figure. Here, for any abstract state $s = (S; \mathit{KB})$, we define $\mathcal{V}(s)$ to be the set containing all variables in Q, for all goals Q_θ or Q_θ^c occurring in S.

$$\begin{tikzcd}
\boxed{S; \mathit{KB}} \arrow[r, dashed, "\gamma"] \arrow[d, "\sigma_\pi"'] & S\gamma \arrow[d, "\theta"] \\
\boxed{S'; \mathit{KB}'} \arrow[r, dashed, "\gamma'"] & S'\gamma'
\end{tikzcd}$$

Consider for example a path π as follows:

$$\begin{tikzcd}
\boxed{(\mathsf{app}(\overline{T_5}, T_7, \overline{T_6}), \mathsf{star}(\overline{T_5}, T_7))_{id}} \arrow[d, "\textsc{Case}"'] \\
\boxed{(\mathsf{app}(\overline{T_5}, T_7, \overline{T_6}), \mathsf{star}(\overline{T_5}, T_7))_{id}^{(4)} \mid (\mathsf{app}(\overline{T_5}, T_7, \overline{T_6}), \mathsf{star}(\overline{T_5}, T_7))_{id}^{(5)}} \arrow[d, "\textsc{Eval}"'] \\
\boxed{\begin{array}{l} \mathsf{star}([\,], \overline{T_{13}})_{\{\overline{T_5}/[\,], T_7/\overline{T_{13}}, \overline{T_6}/\overline{T_{13}}\}} \mid \\ (\mathsf{app}([\,], T_7, \overline{T_{13}}), \mathsf{star}([\,], T_7))^{(5)}_{\{\overline{T_5}/[\,], \overline{T_6}/\overline{T_{13}}, YS/\overline{T_{13}}\}} \end{array}}
\end{tikzcd}$$

In this graph, we only depicted those parts of the substitutions in the labels which concern the variables occurring in states.

Thus, we have $\sigma_\pi = \{\overline{T_5}/[\,], T_7/\overline{T_{13}}, \overline{T_6}/\overline{T_{13}},\ YS/\overline{T_{13}}\}$. Now consider the concretization $S_1 = (\mathsf{app}([\,], X, [1]), \mathsf{star}([\,], X))_{id}$ of π's start node with $\gamma = \{\overline{T_5}/[\,], T_7/X, \overline{T_6}/[1]\}$ and the concretization $S_2 = (\mathsf{star}([\,], [1])_{\{X/[1]\}} \mid (\mathsf{app}([\,], X, [1]), \mathsf{star}([\,], X))_{id}^{(5)})$ of π's end node with $\gamma' = \{T_7/X, \overline{T_{13}}/[1]\}$. Then we have $S_1 \vdash^2_{\{X/[1]\}} S_2$ (i.e., $\theta = \{X/[1]\}$). Now we see that $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = \{\overline{T_5}/[\,], T_7/[1], \overline{T_6}/[1]\} = (\gamma\theta)|_{\mathcal{V}(s)}$.

The following lemma shows that this connection always holds.

Lemma 30 (Concrete/Abstract Substitutions). *Let $\pi = s_1 \dots s_k$ be a path with $s_j \notin \mathit{Inst}(Gr) \cup \mathit{Split}(Gr)$ for all $j \in \{1, \dots, k-1\}$. Let $s_1 = (S_1; \mathit{KB}_1)$, $s_k = (S_k; \mathit{KB}_k)$ and let $S_1\gamma_1 \in \mathcal{CON}(s_1)$ and $S_k\gamma_k \in \mathcal{CON}(s_k)$ be concretizations such that $S_1\gamma_1 \vdash^* S_k\gamma_k$.*[15] *Moreover, let S_1 be of the form Q_σ or Q_σ^c for some query Q, some substitution σ, and some clause c. So $S_1\gamma_1$ is of the form $(Q\gamma_1)_\theta$ or $(Q\gamma_1)_\theta^c$ for some substitution θ. Let n be the number of goals in S_k which are no scope markers and let $\sigma\,\sigma^i$ be the substitution label of the i-th such goal, for all $1 \le i \le n$. Furthermore, let $\theta\,\theta^i$ be the substitution label of the i-th goal in $S_k\gamma_k$ which is no scope marker. Then there is a concretization γ_k' w.r.t. KB_k such that $S_k\gamma_k = S_k\gamma_k'$ and for all $1 \le i \le n$ we have $(\sigma^i\gamma_k')|_{\mathcal{V}(Q)} = (\gamma_1\theta^i)|_{\mathcal{V}(Q)}$. In particular, if π is a connection path, then we have $(\sigma_\pi\gamma_k')|_{\mathcal{V}(Q)} = (\gamma_1\theta_\pi)|_{\mathcal{V}(Q)}$ for $S_1\gamma_1 \vdash^{k-1}_{\theta_\pi} S_k\gamma_k'$.*[16]

Proof. We perform the proof by induction on the length k of the path π. For $k = 1$ we have $s_1 = s_k$, $n = 1$, $\sigma^1 = id = \theta^1$, and $\gamma_1 = \gamma_k$. So we choose $\gamma_k' = \gamma_k$. Then the lemma trivially holds. For $k > 1$ we can assume the lemma holds for paths of length at most $k - 1$.

By Thm. 5, every concrete evaluation $S_1\gamma_1 \vdash^* S_k\gamma_k$ corresponds to the traversal of a path in the symbolic evaluation graph. So $s_k = \mathit{Succ}(s_{k-1})$ for $s_{k-1} = (S_{k-1}, \mathit{KB}_{k-1})$ with $\mathit{KB}_{k-1} = (\mathcal{G}_{k-1}, \mathcal{U}_{k-1})$. Let m be the number of goals in the sequence S_{k-1} which are no scope markers. Let their substitution labels be $\sigma\,\sigma^{i'}$ for $1 \le i \le m$. Let $S_{k-1}\gamma_{k-1} \in \mathcal{CON}(s_{k-1})$ be the predecessor of $S_k\gamma_k$ in the evaluation $S_1\gamma_1 \vdash^* S_k\gamma_k$. Then the goals in $S_{k-1}\gamma_{k-1}$ which are no scope markers have the substitution labels $\theta\,\theta^{i'}$ for $1 \le i \le m$.

By the induction hypothesis, there exists a concretization γ_{k-1}' w.r.t. KB_{k-1} such that $S_{k-1}\gamma_{k-1} = S_{k-1}\gamma_{k-1}'$ and for all $1 \le i \le m$ we have $(\sigma^{i'}\gamma_{k-1}')|_{\mathcal{V}(Q)} = (\gamma_1\theta^{i'})|_{\mathcal{V}(Q)}$.

If s_{k-1} is an Eval node and $s_k = \mathit{Succ}_1(s_{k-1})$, then the sequence S_{k-1} also contains n goals which are no scope markers (i.e., we have $m = n$). Here, the first goal of S_{k-1} has the form $(t, Q')^{h \,:\!-\, B}_{\sigma\sigma^{1'}}$ and the first goal of S_k has the form $(B\sigma', Q'\sigma')_{\sigma\sigma^1}$ where $\sigma' = mgu(t, h)$. Thus, $\sigma^1 = \sigma^{1'}\sigma'$ and for the remaining goals where $2 \le i \le n$, we have $\sigma^i = \sigma^{i'}(\sigma'|_{\mathcal{G}_{k-1}})$. Moreover, the first goal of $S_{k-1}\gamma_{k-1}$ has the form $(t, Q')^{h \,:\!-\, B}_{\theta\theta^{1'}}\gamma_{k-1}$. Thus, $\theta^1 = \theta^{1'}\theta'$ for $\theta' = mgu(t\gamma_{k-1}, h)$. For the remaining goals where $2 \le i \le n$, we

[15] Here, $\vdash^*$ denotes $\vdash_\theta^k$ for an arbitrary $k \ge 0$ and an arbitrary substitution θ.

[16] This follows from $(\sigma^i\gamma_k')|_{\mathcal{V}(Q)} = (\gamma_1\theta^i)|_{\mathcal{V}(Q)}$. The reason is that $\sigma_\pi = \sigma^1$ and $\theta_\pi = \theta^1$.

have $\theta^i = \theta^{i'}$.

Since h does not contain abstract variables and γ'_{k-1} is a concretization, we have $h = h\gamma'_{k-1}$ and, thus, $\theta' = \mathit{mgu}(t\gamma'_{k-1}, h\gamma'_{k-1})$ and $\gamma'_{k-1}\theta'$ is a unifier of t and h. Moreover, we have $\sigma' = \mathit{mgu}(t, h)$. Hence, there must be a substitution ξ with $\sigma'\xi = \gamma'_{k-1}\theta'$.

We define $T\gamma'_k = T\gamma'_{k-1}$ for all $T \in \mathcal{A} \setminus \mathcal{V}(\mathit{Range}(\sigma'))$ and $T\gamma'_k = T\xi$ for all $T \in \mathcal{V}(\mathit{Range}(\sigma'))$.

We now show that for all $i \in \{1, \ldots, n\}$ we have $(\sigma^i\gamma'_k)|_{\mathcal{V}(Q)} = (\gamma_1\theta^i)|_{\mathcal{V}(Q)}$.

For $i = 1$ we have

$$\begin{aligned}
(\sigma^1\gamma'_k)|_{\mathcal{V}(Q)} &= (\sigma^{1'}\sigma'\gamma'_k)|_{\mathcal{V}(Q)}\\
&\overset{(*)}{=} (\sigma^{1'}\gamma'_{k-1}\theta')|_{\mathcal{V}(Q)}\\
&= (\sigma^{1'}\gamma'_{k-1})|_{\mathcal{V}(Q)}\ \theta'|_{\mathcal{V}(\mathcal{V}(Q)\sigma^{1'}\gamma'_{k-1})}\\
&\overset{\text{(Ind. Hyp.)}}{=} (\gamma_1\theta^{1'})|_{\mathcal{V}(Q)}\ \theta'|_{\mathcal{V}(\mathcal{V}(Q)\gamma_1\theta^{1'})}\\
&= (\gamma_1\theta^{1'}\theta')|_{\mathcal{V}(Q)}\\
&= (\gamma\theta^1)|_{\mathcal{V}(Q)}.
\end{aligned}$$

To see the step $(*)$, consider a variable $X \in \mathcal{V}(Q)$. We know that $X \in \mathcal{A}$. Since $\mathcal{V}(\mathit{Range}(\sigma^{1'})) \subseteq \mathcal{A}$, we obtain $\mathcal{V}(X\sigma^{1'}) \subseteq \mathcal{A}$. Moreover, we know that $\mathcal{V}(X\sigma^{1'}) \subseteq \mathit{Dom}(\sigma')$ as σ' replaces all previously occurring variables and has only fresh abstract variables in its range. So for $(*)$, it suffices to show $X'\sigma'\gamma'_k = X'\gamma'_{k-1}\theta'$ for all $X' \in \mathit{Dom}(\sigma')$. Since then all variables in $X'\sigma'$ are from $\mathcal{V}(\mathit{Range}(\sigma'))$, by the definition of γ'_k, we have $X'\sigma'\gamma'_k = X'\sigma'\xi = X'\gamma'_{k-1}\theta'$.

For $i \in \{2, \ldots, n\}$ we have

$$\begin{aligned}
(\sigma^{i}\gamma'_k)|_{\mathcal{V}(Q)} &= (\sigma^{i'}(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k)|_{\mathcal{V}(Q)}\\
&\overset{(**)}{=} (\sigma^{i'}\gamma'_{k-1})|_{\mathcal{V}(Q)}\\
&\overset{\text{(Ind. Hyp.)}}{=} (\gamma_1\theta^{i'})|_{\mathcal{V}(Q)}\\
&= (\gamma_1\theta^{i})|_{\mathcal{V}(Q)}.
\end{aligned}$$

To see the step $(**)$, consider a variable $X \in \mathcal{V}(Q)$. We know that $X \in \mathcal{A}$. Since $\mathcal{V}(\mathit{Range}(\sigma^{i'})) \subseteq \mathcal{A}$, we obtain $\mathcal{V}(X\sigma^{i'}) \subseteq \mathcal{A}$. Moreover, we know that $\mathcal{V}(X\sigma^{i'}) \subseteq \mathit{Dom}(\sigma')$ as σ' replaces all previously occurring variables and has only fresh abstract variables in its range. We perform a case analysis w.r.t. the partition $\mathcal{A}(X\sigma^{i'}) = (\mathcal{A}(X\sigma^{i'}) \cap \mathcal{G}_{k-1}) \uplus (\mathcal{A}(X\sigma^{i'}) \setminus \mathcal{G}_{k-1})$. If $X' \in \mathcal{A}(X\sigma^{i'}) \cap \mathcal{G}_{k-1}$, we have $X'\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k = X'\sigma'\gamma'_k$. As $X' \in \mathit{Dom}(\sigma')$, all variables in $X'\sigma'$ are from $\mathcal{V}(\mathit{Range}(\sigma'))$. So by the definition of γ'_k, we have $X'\sigma'\gamma'_k = X'\sigma'\xi = X'\gamma'_{k-1}\theta'$. As $X' \in \mathcal{G}_{k-1}$, $X'\gamma_{k-1}$ is a ground term and we have $X'\gamma'_{k-1}\theta' = X'\gamma'_{k-1}$. If $X' \in \mathcal{A}(X\sigma^{i'}) \setminus \mathcal{G}_{k-1}$, we have $X'\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k = X'\gamma'_k$. Note that $X' \in \mathit{Dom}(\sigma')$ implies that $X' \notin \mathcal{A}(\mathit{Range}(\sigma'))$. Hence, by the definition of γ'_k we obtain $X'\gamma'_k = X'\gamma'_{k-1}$.

It remains to show that γ'_k is a concretization w.r.t. KB_k and that $S_k\gamma_k = S_k\gamma'_k$. Obviously, we choose the same substitution labels in $S_k\gamma_k$ and $S_k\gamma'_k$. So we disregard substitution labels for the remaining proof. Let S_{suffix_k} be S_k without its first goal. Then we have $S_k\gamma_k = ((B\sigma'\gamma_k, Q'\sigma'\gamma_k) \mid S_{\mathit{suffix}_k}\gamma_k)$ and $S_k\gamma'_k = ((B\sigma'\gamma'_k, Q'\sigma'\gamma'_k) \mid S_{\mathit{suffix}_k}\gamma'_k)$. Recall that $S_{k-1}\gamma_{k-1}$ was the predecessor of $S_k\gamma_k$ in the evaluation $S_1\gamma_1 \vdash^* S_k\gamma_k$ and its first goal was $(t, Q')^{h \,:\!-\, B}\gamma_{k-1}$. Moreover, $\theta' = \mathit{mgu}(t\gamma_{k-1}, h)$. Let $S_{\mathit{suffix}_{k-1}}$ be S_{k-1} without its first goal. Then we have $S_{\mathit{suffix}_k} = S_{\mathit{suffix}_{k-1}}(\sigma'|_{\mathcal{G}_{k-1}})$. Hence, we obtain $B\sigma'\gamma_k = B\theta'$, $Q'\sigma'\gamma_k = Q'\gamma_{k-1}\theta'$, and $S_{\mathit{suffix}_{k-1}}(\sigma'|_{\mathcal{G}_{k-1}})\gamma_k = S_{\mathit{suffix}_{k-1}}\gamma_{k-1}$. To prove $S_k\gamma_k = S_k\gamma'_k$, we therefore have to show $B\sigma'\gamma'_k = B\theta'$, $Q'\sigma'\gamma'_k = Q'\gamma_{k-1}\theta'$, and $S_{\mathit{suffix}_{k-1}}(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k = S_{\mathit{suffix}_{k-1}}\gamma_{k-1}$.

For B we have $B\theta' = B\gamma'_{k-1}\theta'$, as B does not contain any abstract variables and hence, it is not modified by γ'_{k-1}. Moreover, $B\gamma'_{k-1}\theta' = B\sigma'\xi = B\sigma'\gamma'_k$, as $\mathcal{V}(B) \subseteq \mathit{Dom}(\sigma')$.

Since $\mathcal{V}(Q') \subseteq \mathit{Dom}(\sigma')$, in an analogous way we obtain $Q'\gamma_{k-1}\theta' = Q'\sigma'\xi = Q'\sigma'\gamma'_k$.

For any goal $\overline{Q}$ (possibly labeled by a clause c) in $S_{\mathit{suffix}_{k-1}}$, we have to show $\overline{Q}(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k = \overline{Q}\gamma_{k-1}$. Note that since $S_{k-1}\gamma_{k-1} = S_{k-1}\gamma'_{k-1}$ by the induction hypothesis, it suffices to show $\overline{Q}(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k = \overline{Q}\gamma'_{k-1}$. We perform a case analysis w.r.t. the partition $\mathcal{A}(\overline{Q}) = (\mathcal{A}(\overline{Q}) \cap \mathcal{G}_{k-1}) \uplus (\mathcal{A}(\overline{Q}) \setminus \mathcal{G}_{k-1})$. If $X \in \mathcal{A}(\overline{Q}) \cap \mathcal{G}_{k-1}$, we have $X(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k = X\sigma'\gamma'_k$. As σ' renames all previously occurring variables, we have $X' \in \mathit{Dom}(\sigma')$ and thus, all variables occurring in $X\sigma'$ are from $\mathit{Range}(\sigma')$. So by the definition of γ'_k, we have $X\sigma'\gamma'_k = X\sigma'\xi = X\gamma'_{k-1}\theta'$. As $X \in \mathcal{G}_{k-1}$, γ'_{k-1} maps X to a ground term and thus, $X\gamma'_{k-1}\theta' = X\gamma'_{k-1}$. If $X \in \mathcal{A}(\overline{Q}) \setminus \mathcal{G}_{k-1}$, then we have $X(\sigma'|_{\mathcal{G}_{k-1}})\gamma'_k = X\gamma'_k$, as $X \notin \mathcal{G}_{k-1}$. Since the range of σ' only contains fresh variables we have $X \notin \mathcal{A}(\mathit{Range}(\sigma'))$. So by the definition of γ'_k, we have $X\gamma'_k = X\gamma'_{k-1}$.

Finally, we need to show that γ'_k is a concretization w.r.t. $\mathit{KB}_k = (\mathcal{G}_k, \mathcal{U}_k)$, i.e., that $\mathit{Dom}(\gamma'_k) = \mathcal{A}$, $\mathcal{V}(\mathit{Range}(\gamma'_k)) \subseteq \mathcal{N}$, $\mathcal{V}(\mathit{Range}(\gamma'_k|_{\mathcal{G}_k})) = \varnothing$, and $\bigwedge_{(s,t)\in\mathcal{U}'} s\gamma'_k \not\sim t\gamma'_k$.

Obviously, we defined γ'_k in such a way that $\mathit{Dom}(\gamma'_k) = \mathcal{A}$.

We now show that $\mathcal{V}(\mathit{Range}(\gamma'_k)) \subseteq \mathcal{N}$. In other words, for all $T \in \mathcal{A}$, we show that $\mathcal{V}(T\gamma'_k) \subseteq \mathcal{N}$. To this end, we perform a case analysis w.r.t. $\mathcal{A} = \mathcal{A}(\mathit{Range}(\sigma')) \uplus (\mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\sigma')))$. For $T \in \mathcal{A}(\mathit{Range}(\sigma'))$ we have $\mathcal{V}(T\gamma'_k) = \mathcal{V}(T\xi)$ by the definition of γ'_k. Assume there is a $T' \in \mathcal{A} \cap \mathcal{V}(T\xi)$. As $T \in \mathcal{A}(\mathit{Range}(\sigma'))$, there is an $X \in \mathit{Dom}(\sigma')$ with $T \in \mathcal{V}(X\sigma')$ and, thus, $T' \in \mathcal{V}(X\sigma'\xi) = \mathcal{V}(X\gamma'_{k-1}\theta')$. However, we have $\mathcal{V}(X\gamma'_{k-1}) \subseteq \mathcal{N}$ and $\mathcal{V}(\mathit{Range}(\theta')) \subseteq \mathcal{N}$, which leads to a contradiction. Thus, we must have $\mathcal{V}(T\gamma'_k) = \mathcal{V}(T\xi) \subseteq \mathcal{N}$. For $T \in \mathcal{A} \setminus \mathcal{A}(\mathit{Range}(\sigma'))$ we have $\mathcal{V}(T\gamma'_k) = \mathcal{V}(T\gamma'_{k-1})$ by the definition of γ'_k and $\mathcal{V}(T\gamma'_{k-1}) \subseteq \mathcal{N}$, since γ'_{k-1} is a concretization.

To show that $\mathcal{V}(\mathit{Range}(\gamma'_k|_{\mathcal{G}_k})) = \varnothing$, let $T \in \mathcal{G}_k = \mathcal{A}(\mathit{Range}(\sigma'|_{\mathcal{G}_{k-1}}))$. Our goal is to show that $\mathcal{V}(T\gamma'_k) = \varnothing$. Recall that $\mathcal{V}(T\gamma'_k) = \mathcal{V}(T\xi)$ for such T. Moreover, there must be a $T' \in \mathit{Dom}(\sigma'|_{\mathcal{G}_{k-1}})$ with $T \in \mathcal{V}(T'\sigma')$. Hence, it suffices to show that $\mathcal{V}(T\gamma'_k) = \mathcal{V}(T\xi) \subseteq \mathcal{V}(T'\sigma'\xi) = \varnothing$. By the definition of ξ, we have $\mathcal{V}(T'\sigma'\xi) = \mathcal{V}(T'\gamma'_{k-1}\theta')$ and $\mathcal{V}(T'\gamma'_{k-1}\theta') = \varnothing$ holds since $T' \in \mathit{Dom}(\sigma'|_{\mathcal{G}_{k-1}}) \subseteq \mathcal{G}_{k-1}$ and γ'_{k-1} is a concretization w.r.t. $(\mathcal{G}_{k-1}, \mathcal{U}_{k-1})$.

Finally, we show that $\bigwedge_{(s,t)\in\mathcal{U}_k} s\gamma'_k \not\sim t\gamma'_k$, where $\mathcal{U}_k = \mathcal{U}_{k-1}\sigma'|_{\mathcal{G}_{k-1}}$. Note that $\bigwedge_{(s,t)\in\mathcal{U}_{k-1}\sigma'|_{\mathcal{G}_{k-1}}} s\gamma'_k \not\sim t\gamma'_k$ holds iff $\bigwedge_{(s,t)\in\mathcal{U}_{k-1}} s\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k \not\sim t\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k$ holds, which in turn holds iff $\bigwedge_{(s,t)\in\mathcal{U}_{k-1}} s\gamma'_{k-1} \not\sim t\gamma'_{k-1}$ holds. (The truth of the last statement follows from the fact that γ'_{k-1} is a concretization w.r.t. $(\mathcal{G}_{k-1}, \mathcal{U}_{k-1})$.) The reason for the last equivalence above is that $T\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k = T\gamma'_{k-1}$ for all $T \in \mathcal{V}(\mathcal{U}_{k-1})$. For non-abstract variables $T \in \mathcal{N}$, this is clear since they are not contained in $\mathcal{G}_{k-1}$, and thus they are not modified by $\sigma'|_{\mathcal{G}_{k-1}}$, γ'_k, or γ'_{k-1}. To see why $T\sigma'|_{\mathcal{G}}\gamma'_k = T\gamma'_{k-1}$

holds for all $T \in \mathcal{A}(\mathcal{U}_{k-1})$, note that $T \notin \mathcal{V}(\mathit{Range}(\sigma'))$ as $\mathcal{V}(\sigma'(X))$ only contains fresh abstract variables for all previously occurring $X \in \mathcal{V}$. Now consider the partition $\mathcal{A}(\mathcal{U}_{k-1}) = (\mathcal{A}(\mathcal{U}_{k-1}) \setminus \mathcal{G}_{k-1}) \uplus (\mathcal{A}(\mathcal{U}_{k-1}) \cap \mathcal{G}_{k-1})$. If $T \in \mathcal{A}(\mathcal{U}_{k-1}) \setminus \mathcal{G}_{k-1}$, we have $T\gamma'_{k-1} = T\gamma'_k$ by the definition of γ'_k as $T \notin \mathcal{V}(\mathit{Range}(\sigma'))$, and $T\gamma'_k = T\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k$ since $T \notin \mathit{Dom}(\sigma'|_{\mathcal{G}_{k-1}})$. If $T \in \mathcal{A}(\mathcal{U}_{k-1}) \cap \mathcal{G}_{k-1}$, we have $T\gamma'_{k-1} = T\gamma'_{k-1}\theta'$, since $T \in \mathcal{G}_{k-1}$. Moreover, $T\gamma'_{k-1}\theta' = T\sigma'\xi$ by the definition of ξ, and $T\sigma'\xi = T\sigma'|_{\mathcal{G}_{k-1}}\xi$ since $T \in \mathcal{G}_{k-1}$. As $\mathcal{V}(\mathit{Range}(\sigma'|_{\mathcal{G}_{k-1}})) \subseteq \mathcal{A}$, by the definition of γ'_k we obtain $T\sigma'|_{\mathcal{G}_{k-1}}\xi = T\sigma'|_{\mathcal{G}_{k-1}}\gamma'_k$.

Now we regard all other cases (where $s_k = \mathit{Succ}(s_{k-1})$, but s_k is not the first successor of an EVAL node).

We know that the substitution labels are neither changed in the last step of the abstract evaluation (from s_{k-1} to s_k) nor in the last step of the concrete evaluation (from $S_{k-1}\gamma_{k-1}$ to $S_k\gamma_k$). Likewise, the variables are not instantiated by any substitution. Thus, we can use the same concretization for s_k as for s_{k-1} and define $\gamma'_k = \gamma'_{k-1} = \gamma_k$. Moreover, the substitution labels of the goals are also not changed when going from s_{k-1} to s_k. Thus, we have $\sigma^i = \sigma^{i'}$. The same holds for the concrete evaluation when going from $S_{k-1}\gamma_{k-1}$ to $S_k\gamma_k$. Thus, we have $\theta^i = \theta^{i'}$. The only exception is that the first goal of s_{k-1} resp. S_{k-1} may have been removed in this evaluation step. In that case, while s_{k-1} resp. S_{k-1} had m goals that were no scope markers, then s_k resp. S_k only have $n = m - 1$ such goals. In that case, we have $\sigma^i = \sigma^{i+1'}$ and $\theta^i = \theta^{i+1'}$. Hence, in that case we obtain

$$\begin{aligned}
(\sigma^i\gamma'_k)|_{\mathcal{V}(Q)} &= (\sigma^{i+1'}\gamma'_k)|_{\mathcal{V}(Q)}\\
&= (\sigma^{i+1'}\gamma'_{k-1})|_{\mathcal{V}(Q)}\\
&\overset{\text{(Ind. Hyp.)}}{=} (\gamma_1\theta^{i+1'})|_{\mathcal{V}(Q)}\\
&= (\gamma_1\theta_i)|_{\mathcal{V}(Q)}
\end{aligned}$$

for all $i \in \{1,\dots,n\}$. Otherwise we have $n = m$ and

$$\begin{aligned}
(\sigma^i\gamma'_k)|_{\mathcal{V}(Q)} &= (\sigma^{i'}\gamma'_k)|_{\mathcal{V}(Q)}\\
&= (\sigma^{i'}\gamma'_{k-1})|_{\mathcal{V}(Q)}\\
&\overset{\text{(Ind. Hyp.)}}{=} (\gamma_1\theta^{i'})|_{\mathcal{V}(Q)}\\
&= (\gamma_1\theta^i)|_{\mathcal{V}(Q)}
\end{aligned}$$

for all $i \in \{1,\dots,n\}$. $\square$

The proof of Thm. 16 uses induction on the length of the successful evaluation of $S\gamma$ to a state starting with $\square$. If this evaluation corresponds to a connection path, Thm. 16 follows by Thm. 5 and the induction hypothesis. If s is a SPLIT node, both the evaluation of the first and the second successor of s to states starting with $\square$ are shorter than the complete evaluation. Thus we can use the induction hypothesis for both successors and Thm. 16 follows by Thm. 10. If s is an INST node, we know by Thm. 8 that we can simulate the same evaluation from its successor. Note that there may not be cycles in an evaluation graph which consist of INST edges only. So there is a path of finitely many INST edges from s to the next state s' that is not an INST node. Since the theorem holds for s', it also holds for s.

THEOREM 16 (TRS SIMULATES SEMANTICS). *Let $s = (S; KB)$ be a start node of a connection path or a* SPLIT *node in a graph Gr, $S\gamma \in \mathcal{CON}(s)$, and let θ be an answer substitution for $S\gamma$. Then $enc^{in}(s)\gamma \xrightarrow{i}^{+}_{\mathcal{R}(Gr)} enc^{out}(s)\gamma\theta$.*

PROOF. We show the theorem by induction on k. We must have $k > 0$ since a state starting with $\square$ cannot be a start node of a connection path or a SPLIT node.

If $k = 1$, then we have $S = (!_m)_{\sigma'}$ for some $m \in \mathbb{N}$ or $S = (t)^c_{\sigma'}$ for some term t and some clause c. In the first case, we applied the CUT rule to s and reach the state $s' = (\square_{\sigma'}; KB)$. So we have a connection path π from s to s'. Moreover, the answer substitution θ is empty. Thus, we trivially have that $enc^{in}(s)\gamma \xrightarrow{i}_{\mathcal{R}(Gr)} enc^{out}(s)\gamma$ as σ_π is the identity. This is an innermost rewrite step, as the function symbols introduced in the TRSs are fresh and do not occur in the range of γ. In the second case, c must be a fact since we reach a success in the next step. Moreover, by Thm. 5 the EVAL rule must have been applied to s. If we had applied the INST rule, s would not be a start node of a connection path. The SPLIT rule cannot be applied since $(t)^c$ is no sequence of terms and for all other rules except EVAL, the successor of $((t)^c_{\sigma'})\gamma$ would not be represented by at least one of the successors of s. So the first successor of s is $s' = (\square_{\sigma'\sigma}; KB')$ with $\sigma = mgu(t, h)$ where h is the head of c. Thus, we have the rule $enc^{in}(s)\sigma \to enc^{out}(s)\sigma$ in $\mathcal{R}(Gr)$. The answer substitution for $S\gamma$ is $\theta = mgu(t\gamma, h)$. Since γ is a concretization and h does not contain abstract variables, we have $h\gamma = h$. Hence, we obtain $mgu(t\gamma, h\gamma) = \theta$ and thus, $\gamma\theta$ is a unifier of t and h. So there must be a substitution ξ with $\sigma\xi = \gamma\theta$. Furthermore, $\mathcal{V}(enc^{in}(s)\gamma) = \varnothing$ as $enc^{in}(s)$ only contains abstract variables which must be replaced by ground terms by γ. Thus, we have $enc^{in}(s)\gamma = enc^{in}(s)\gamma\theta = enc^{in}(s)\sigma\xi \xrightarrow{i}_{\mathcal{R}(Gr)} enc^{out}(s)\sigma\xi = enc^{out}(s)\gamma\theta$. Again, the rewrite step is innermost as the function symbols introduced in the TRSs are fresh and do not occur in the range of ξ.

Let $k > 1$. As long as we do not apply the INST or SPLIT rule, the concrete evaluation of $S\gamma$ directly corresponds to a path in the symbolic evaluation graph by Thm. 5.

If we do not apply the INST or SPLIT rule and we do not traverse a successor of an INST node while simulating the concrete evaluation, this simulation must end in a SUC node and, thus, we have a connection path π from s to a state $s' = (\square_{\sigma'} \mid S'; KB')$ which simulates all k steps of the concrete evaluation. In particular, for the concrete final state we have $(\square_{\theta'\theta} \mid S_{\mathit{suffix}}) = (\square_{\sigma'} \mid S')\gamma' \in \mathcal{CON}(s')$. Hence, we have the rule $enc^{in}(s)\sigma_\pi \to enc^{out}(s)\sigma_\pi$ in $\mathcal{R}(Gr)$. By Lemma 30, $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta)|_{\mathcal{V}(s)}$ and we obtain $enc^{in}(s)\gamma = enc^{in}(s)\gamma\theta = enc^{in}(s)\sigma_\pi\gamma' \xrightarrow{i}_{\mathcal{R}(Gr)} enc^{out}(s)\sigma_\pi\gamma' = enc^{out}(s)\gamma\theta$ by the same reasoning as in the base case of the induction.

If we apply the INST or SPLIT rule (but not to the first node s) or if we reach a successor of an INST node in more than zero steps following the concrete evaluation through the graph, then we have a connection path from s to some node s' which is no SUC node. Thus, we have the rules $enc^{in}(s)\sigma_\pi \to \mathsf{u}_{s,s'}(enc^{in}(s'), \mathcal{V}(enc^{in}(s)\sigma_\pi))$ and $\mathsf{u}_{s,s'}(enc^{out}(s'), \mathcal{V}(enc^{in}(s)\sigma_\pi)) \to enc^{out}(s)\sigma_\pi$ in $\mathcal{R}(Gr)$. As both INST and SPLIT only work on states with one single goal, we know that s' only has one goal, i.e., $s' = (Q_{\sigma'\sigma_\pi}; KB')$ (where Q may additionally be labeled by a clause c). Hence, we have $S\gamma \vdash^{k'}_{\theta''} (Q\gamma')_{\theta'\theta''} \in \mathcal{CON}(s')$ with $k > k' > 0$. If s' is an INST node, there is a node s'' in Gr which is reachable from s' by INST edges only and $(Q\gamma')_{\theta'\theta''} \in \mathcal{CON}(s'')$. So w.l.o.g., we can assume that s' is a

SPLIT node or a successor of an INST node which starts again a connection path (in this latter case, we might have to take s'' for s'). As $(Q\gamma')_{\sigma'\sigma_\pi} \vdash^{k-k'}_{\theta'''} (\Box_{\theta'\theta''\theta'''} \mid S_{suffix})$ for some substitution θ''' with $\theta = \theta''\theta'''$, we can use the induction hypothesis to obtain $enc^{in}(s')\gamma' \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} enc^{out}(s')\gamma'\theta'''$. By Lemma 30, we know that $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta'')|_{\mathcal{V}(s)}$, where π is the connection path from s to s'. Thus, we obtain

$$\begin{array}{lll} enc^{in}(s)\gamma & = & enc^{in}(s)\gamma\theta'' \\ & = & enc^{in}(s)\sigma_\pi\gamma' \\ & \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} & \mathsf{u}_{s,s'}(enc^{in}(s')\gamma', \mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma') \\ & \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} & \mathsf{u}_{s,s'}(enc^{out}(s')\gamma'\theta''', \mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma') \\ & \overset{(*)}{=} & \mathsf{u}_{s,s'}(enc^{out}(s')\gamma'\theta''', \\ & & \mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma'\theta''') \\ & \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} & enc^{out}(s)\sigma_\pi\gamma'\theta''' \\ & = & enc^{out}(s)\gamma\theta''\theta''' \\ & = & enc^{out}(s)\gamma\theta. \end{array}$$

This is an innermost reduction, as the defined function symbols in the TRS are fresh and do not occur in the ranges of the involved substitutions. For the step $(*)$, note that $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta'')|_{\mathcal{V}(s)}$ and γ is a concretization w.r.t. $KB = (\mathcal{G}, \mathcal{U})$. In particular, it instantiates all variables in $\mathcal{G}$ by ground terms and $enc^{in}(s)$ only contains variables from $\mathcal{G}$. So $\mathcal{V}(\mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma') = \varnothing$ and, hence, $\mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma' = \mathcal{V}(enc^{in}(s)\sigma_\pi)\gamma'\theta'''$.

We know that s cannot be an INST node as it must be a start node of a connection path or a SPLIT node.

If s is a SPLIT node, we have $S = (t, Q)_{\sigma'}$. Let $s_1 = (t_{id}; KB)$ be its first successor and $s_2 = ((Q\delta)_\delta; KB')$ its second successor. Then we have the rules $enc^{in}(s) \to \mathsf{u}_{s,s_1}(enc^{in}(s_1), \mathcal{V}(enc^{in}(s)))$, $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))) \to \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta))$, and $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \to enc^{out}(s)\delta$ in $\mathcal{R}(Gr)$. So we directly obtain $enc^{in}(s)\gamma \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathsf{u}_{s,s_1}(enc^{in}(s_1)\gamma, \mathcal{V}(enc^{in}(s))\gamma)$. Moreover, as the concrete evaluation of $(t\gamma, Q\gamma)_{\sigma'}$ reaches a state $(\Box_{\theta'\theta} \mid S_{suffix})$, there must be substitutions θ'' and θ''' such that $(t\gamma)_{id} \vdash^{k'}_{\theta''} (\Box_{\theta''} \mid S')$ and $(Q\gamma\theta'')_{\theta''} \vdash^{k''}_{\theta'''} (\Box_{\theta''\theta'''} \mid S_{suffix})$ where $0 < k' < k$, $0 < k'' \leq k - k'$, and $\theta = \theta''\theta'''$. Since s_1 must be (an instance of) a start node of a connection path, we can use the induction hypothesis to obtain $enc^{in}(s_1)\gamma \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} enc^{out}(s_1)\gamma\theta''$, i.e., $\mathsf{u}_{s,s_1}(enc^{in}(s_1)\gamma, \mathcal{V}(enc^{in}(s))\gamma) \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma)$. Furthermore, we have $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma) = \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma\theta'')$ as γ is a concretization w.r.t. KB and $enc^{in}(s)$ only contains variables from $\mathcal{G}$ for $KB = (\mathcal{G}, \mathcal{U})$. By Thm. 10 there is a concretization γ' w.r.t. KB' such that $Q\gamma\theta'' = Q\delta\gamma'$ and $t\gamma\theta'' = t\delta\gamma'$. Moreover, the rule

$$\begin{array}{c} \mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))) \\ \to \\ \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \end{array}$$

is the same as the rule

$$\begin{array}{c} \mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))\delta) \\ \to \\ \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta) \end{array}$$

since $enc^{in}(s)$ only contains variables from $\mathcal{G}$ and $\mathcal{G} \cap Dom(\delta) = \varnothing$. Hence, we obtain $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma\theta'') \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathsf{u}_{s_1,s_2}(enc^{in}(s_2)\gamma', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma')$. This is an innermost rewrite step, as the defined function symbols in the TRS are fresh and do not occur in the ranges of the involved substitutions. As s_2 must also be (an instance of) a start node of a connection path or a SPLIT node, we can again use the induction hypothesis to obtain $enc^{in}(s_2)\gamma' \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} enc^{out}(s_2)\gamma'\theta'''$, i.e., we have:

$$\begin{array}{c} \mathsf{u}_{s_1,s_2}(enc^{in}(s_2)\gamma', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma') \\ \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} \\ \mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma') \end{array}$$

We also know that the rule $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \to enc^{out}(s)\delta$ is the same as the rule $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta) \to enc^{out}(s)\delta$ since, again, δ does not instantiate any variables in $enc^{in}(s)$. Moreover, we know that

$$\begin{array}{lc} (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma' & = \\ (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma'\theta''' & \end{array}$$

because we have $\delta\gamma' = \gamma\theta''$ on all variables in $enc^{in}(s)$ and $enc^{out}(s_1)$ and by the groundness analysis we are guaranteed to instantiate all these variables by ground terms using $\gamma\theta''$. This yields $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma') = \mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma'\theta''') \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} enc^{out}(s)\delta\gamma'\theta''' = enc^{out}(s)\gamma\theta''\theta''' = enc^{out}(s)\gamma\theta$. Altogether, we have $enc^{in}(s)\gamma \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} enc^{out}(s)\gamma\theta$ which concludes the proof. □

The proof of Thm. 17 is performed by showing the following property. Suppose that there is an abstract state $s = (S; KB)$ which is a SPLIT node or the start node of a connection path, and that its concretization $S\gamma$ starts an infinite evaluation. Then there is a prefix of that evaluation which reaches a concrete state $(S'\gamma' \mid S_{suffix})$ such that $S'\gamma'$ again starts an infinite evaluation and it is a concretization of an abstract state $s' = (S'; KB')$ that is a SPLIT node or the start node of a connection path. Moreover, this prefix of the evaluation can be simulated by a rewrite sequence with the TRS corresponding to the evaluation graph. The start term of that rewrite sequence is $enc^{in}(s)\gamma$ and the final term of the rewrite sequence contains $enc^{in}(s')\gamma'$. Thus, by repeating this construction, one obtains an infinite rewrite sequence. To prove the above property one uses Thm. 16 if s is a SPLIT node and Thm. 5 if s starts a connection path.

THEOREM 17 (SOUNDNESS OF TERMINATION ANALYSIS). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}^{\mathsf{p}}_m$. If the TRS $\mathcal{R}(Gr)$ is innermost terminating, then there is no infinite evaluation starting with any query from $\mathcal{Q}^{\mathsf{p}}_m$. Thus, all these queries are terminating w.r.t. the program $\mathcal{P}$.*

PROOF. We show the following proposition (*): Let $S\gamma$ be a concretization of an abstract state $s = (S; KB)$ which is a SPLIT node or a start node of a connection path, where $S\gamma$ has an infinite derivation. Then there is an evaluation $S\gamma \vdash^{+} (S'\gamma' \mid S_{suffix})$ such that

(a) there is an abstract state $s' = (S'; KB')$ in Gr with $S'\gamma' \in \mathcal{CON}(s')$,

(b) $S'\gamma'$ has an infinite evaluation,

(c) s' is a Split node, a start node of a connection path, or an Inst node, and

(d) $enc^{in}(s)\gamma \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}(Gr)} C[enc^{in}(s')\gamma']$ for some context C.

The proposition (*) obviously implies the theorem. If there were an infinite evaluation starting with a query from $\mathcal{Q}^{\mathsf{p}}_m$, then this query has the form $S\gamma$ where $S\gamma \in \mathcal{CON}(s)$ for the root node s of Gr. Since s is the start node of a connection path, by (*) we have $S\gamma \vdash^{+} (S'\gamma' \mid S_{suffix})$, where $S'\gamma' \in \mathcal{CON}(s')$ for a state s' in Gr by (a). By (c), s' is a Split node, a start node of a connection path, or an Inst node. In the latter case, since symbolic evaluation graphs are finite and may not contain cycles consisting only of Inst edges, by following the path of Inst edges originating in s', we reach a node $s'' = (S''; KB'')$ which is again a Split node or a start node of a connection path. Thus, $enc^{in}(s') = enc^{in}(s'')\mu$, where μ results from composing the substitutions on these Inst edges (we define $s' = s''$ and $\mu = id$ in the case where s' itself is already a Split node or a start node of a connection path). Let $\gamma'' = \mu\gamma'$. Then by (d), there is an innermost rewrite sequence from $enc^{in}(s)\gamma$ to a term containing $enc^{in}(s'')\gamma''$. Note that $S'\gamma' = S''\gamma''$ has an infinite evaluation by (b) and that it is also a concretization of s'' by Thm. 8. Hence, we can continue the above construction and obtain an innermost rewrite sequence from $enc^{in}(s'')\gamma''$ to a term containing $enc^{in}(s'''')\gamma''''$, etc.

Now we prove the proposition (*).

If s is a Split node, then we know that S contains a single goal resulting from a query (t, Q). Let δ be the substitution associated to s. Let V contain all abstract variables occurring in s. Then $\delta|_{V\setminus\mathcal{G}}$ must be injective as the variables in its range are fresh. So there exists a substitution δ^{-1} with $\delta^{-1}(T') = T$ if there is a $T \in V \setminus \mathcal{G}$ with $\delta(T) = T'$ and $\delta^{-1}(T') = T'$ for all other variables T'. There are two cases.

If $t\gamma$ already has an infinite evaluation, then we let $s' = (t_{id}; KB)$ be the first successor of s. Obviously, s' is a start node of a connection path or an Inst node (i.e., (c) holds). By defining $\gamma' = \gamma$, we directly have that (a) and (b) hold as well. Moreover, $\mathcal{R}(Gr)$ contains the rule $enc^{in}(s) \to \mathsf{u}(enc^{in}(s'), \mathcal{V}(enc^{in}(s)))$ for some fresh function symbol u. Thus, $enc^{in}(s)\gamma \to_{\mathcal{R}(Gr)} C[enc^{in}(s')\gamma]$. Note that this is an innermost rewrite step, since the function symbols f^{in}_s and u introduced in the TRSs are fresh and do not occur in the range of γ'.

Otherwise, if $t\gamma$ does not have an infinite evaluation, then there must be some answer substitution θ of $(t\gamma)_{id}$ with $(t\gamma)_{id} \vdash^{+}_{\theta} ((Q\gamma\theta)_{\theta} \mid S_{suffix})$ and $Q\gamma\theta$ has an infinite evaluation. Let $s' = ((Q\delta)_{\delta}; KB')$ be the second successor of s. We define $T\gamma' = T\gamma\theta$ for $T \in \mathcal{A} \setminus \mathcal{A}(Range(\delta|_{V\setminus\mathcal{G}}))$ and $T\gamma' = T\delta^{-1}\gamma\theta'$ for $T \in \mathcal{A}(Range(\delta|_{V\setminus\mathcal{G}}))$. We have $KB' = (\mathcal{G}', \mathcal{U}') = (\mathcal{G} \cup NextG(t, \mathcal{G})\delta, \mathcal{U}\delta)$. We need to show that γ' is a concretization w.r.t. KB' and that $Q\gamma\theta' = Q\delta\gamma'$. This proof is completely analogous to the proof of Thm. 10. Thus, (a) and (b) hold. Moreover, s' is clearly a Split node, a start node of a connection path, or an Inst node (i.e., (c) also holds). Finally, by Thm. 16 we know that $enc^{in}(Succ_1(s))\gamma \to^{+}_{\mathcal{R}(Gr)} enc^{out}(Succ_1(s))\gamma\theta$. Moreover, $\mathcal{R}(Gr)$ contains the rules

$$enc^{in}(s) \to \mathsf{u}_1(enc^{in}(Succ_1(s)), \mathcal{V}(enc^{in}(s)))$$

and

$$\begin{array}{c}\mathsf{u}_1(enc^{out}(Succ_1(s))\delta, \mathcal{V}(enc^{in}(s))) \\ \to \\ \mathsf{u}_2(enc^{in}(s'), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(Succ_1(s))\delta)).\end{array}$$

We have

$$\begin{array}{rll} enc^{in}(s)\gamma & \to_{\mathcal{R}(Gr)} & \mathsf{u}_1(enc^{in}(Succ_1(s))\gamma, \mathcal{V}(enc^{in}(s))\gamma) \\ & \to^{+}_{\mathcal{R}(Gr)} & \mathsf{u}_1(enc^{out}(Succ_1(s))\gamma\theta, \mathcal{V}(enc^{in}(s))\gamma). \end{array}$$

Note that γ instantiates all variables of $enc^{in}(s)$ to ground terms. Thus, the term obtained above is identical to $\mathsf{u}_1(enc^{out}(Succ_1(s))\gamma\theta, \mathcal{V}(enc^{in}(s))\gamma\theta)$. Hence, by instantiating the rule $\mathsf{u}_1(\ldots) \to \mathsf{u}_2(\ldots)$ with γ' we can rewrite this term to $C[enc^{in}(s')\gamma']$. As before, all rewrite steps are innermost, i.e., (c) holds as well.

If s is a start node of a connection path, the beginning of $S\gamma$'s infinite evaluation corresponds to the traversal of a connection path in Gr by Thm. 5. Note that we can follow this connection path until we reach an Inst or Split node or the successor of an Inst node, and we let $s' = (S'; KB')$ be the end node of this connection path π (i.e., we do not stop at Suc nodes). By Thm. 5, there is an $S'\gamma' \in \mathcal{CON}(s')$ with $S\gamma \vdash^{+}_{\theta} S'\gamma'$ for some substitution θ and $S'\gamma'$ again has an infinite evaluation (i.e., (a) and (b) hold). Moreover, $\mathcal{R}(Gr)$ contains the rule $enc^{in}(s)\sigma_\pi \to \mathsf{u}(enc^{in}(s'), \mathcal{V}(enc^{in}(s)\sigma_\pi))$ for some fresh function symbol u. By Def. 7 and 9, we know that s only contains one single goal. Thus, by Lemma 30 we obtain $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta)|_{\mathcal{V}(s)}$. Furthermore, we have $\mathcal{V}(enc^{in}(s)\gamma) = \varnothing$ and, thus, $enc^{in}(s)\gamma = enc^{in}(s)\gamma\theta = enc^{in}(s)\sigma_\pi\gamma' \to_{\mathcal{R}(Gr)} C[enc^{in}(s')\gamma']$. Again this is an innermost rewrite step, since the function symbols f^{in}_s and u introduced in the TRSs are fresh and do not occur in the ranges of σ_π or γ'. Thus, (d) holds. Moreover, s' is either a Split node, the successor of an Inst node, or an Inst node (i.e., (c) holds as well). □

To prove Thm. 23, we need the following Lemmas 31 and 33. To prove Lemma 33, we need an additional Lemma 32. Moreover, Lemma 33 will also be needed to prove Thm. 27. Thus, it is already generalized to graphs containing multiplicative Split nodes.

Lemma 31 (Start Queries and Basic Terms). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root $s = (\mathsf{p}(T_1, \ldots, T_n)_{id}; (\{T_i \mid m(\mathsf{p}, i) = in\}, \varnothing))$ is the initial state corresponding to $\mathcal{Q}^{\mathsf{p}}_m$. For each concretization γ w.r.t. $(\{T_i \mid m(\mathsf{p}, i) = in\}, \varnothing)$, we have*[17] *$|\mathsf{p}(T_1, \ldots, T_n)\gamma|_m = |enc^{in}(s)\gamma|$ and $enc^{in}(s)\gamma$ is a basic term w.r.t. $\mathcal{R}(Gr)$.*

Proof. Let $\mathsf{p}(T_1, \ldots, T_n)\gamma = \mathsf{p}(t_1, \ldots, t_n)_\theta \in \mathcal{CON}(s)$. Moreover, let $k = |\{i \in \{1, \ldots, n\} \mid m(\mathsf{p}, i) = in\}|$ and let $\{i_1, \ldots, i_k\} = \{i \in \{1, \ldots, n\} \mid m(\mathsf{p}, i) = in\}$ with $i_j < i_{j+1}$ for all $j \in \{1, \ldots, k-1\}$. Then we have $|\mathsf{p}(t_1, \ldots, t_n)_\theta|_m = 1 + \Sigma_{i \in \{i \mid 1 \le i \le n, m(\mathsf{p},i)=in\}} |t_i| = 1 + \Sigma_{j \in \{1,\ldots,k\}} |t_{i_j}| = |f^{in}_s(t_{i_1}, \ldots, t_{i_k})| = |f^{in}_s(T_{i_1}, \ldots, T_{i_k})\gamma| = |enc^{in}(s)\gamma|$. Furthermore, $enc^{in}(s)\gamma = f^{in}_s(T_{i_1}, \ldots, T_{i_k})\gamma$ must be a basic term w.r.t. $\mathcal{R}(Gr)$ as all defined function symbols in $\mathcal{R}(Gr)$ are fresh and, thus, they do not occur in the range of γ. □

[17]Here, we simply ignore the substitution labels when computing $|\mathsf{p}(T_1, \ldots, T_n)\gamma|_m$.

Lemma 32 (Complexity Approximating Answer). *Let Gr be a symbolic evaluation graph and $g = |Gr|$ be the number of nodes in Gr. Let $S\gamma$ be a concretization of an abstract state $s = (S; KB)$ which is a start node of a connection path or a* Split *node. Moreover, let the longest evaluation of $S\gamma$ be finite and compute exactly one answer substitution θ, i.e., it is of the form $S\gamma \vdash^{\ell_1} (\Box_{\theta'\theta} \mid S_{\mathit{suffix}}) \vdash^{\ell_2} \varepsilon$. Then we have $\mathit{enc}^{\mathit{in}}(s)\gamma \xrightarrow{\mathsf{i}}{}^{r}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s)\gamma\theta$ such that $g \cdot r \geq \ell_1$.*

Proof. We show this proposition by induction on ℓ_1. We must have $\ell_1 > 0$ since a state starting with $\Box$ cannot be a start node of a connection path or a Split node.

If $\ell_1 = 1$, then we have $S = (!_m)_{\sigma'}$ for some $m \in \mathbb{N}$ or $S = (t)^c_{\sigma'}$ for some term t and some clause c. In the first case, we applied the Cut rule to s and reach the state $s' = (\Box_{\sigma'}; KB)$. So we have a connection path from s to s'. Moreover, the answer substitution θ is empty. Thus, we trivially have that $\mathit{enc}^{\mathit{in}}(s)\gamma \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s)\gamma$ (as the defined function symbols in the TRSs are fresh and do not occur in the range of γ), $r = 1$, and $g \geq 1$. In the second case, c must be a fact since we reach a success in the next step. Moreover, by Thm. 5 we know that the Eval rule must have been applied to s. If we would have applied the Inst rule, s would not be a start node of a connection path. The Split rule cannot be applied since $(t)^c$ is no sequence of terms and for all other rules except Eval, the successor of $((t)^c_{\sigma'})\gamma$ would not be represented by at least one of the successors of s. So the first successor of s is $s' = (\Box_{\sigma'\sigma}; KB')$ with $\sigma = \mathit{mgu}(t, h)$ where h is the head of c. We know that the answer substitution is $\theta = \mathit{mgu}(t\gamma, h)$ and we have the rule $\mathit{enc}^{\mathit{in}}(s)\sigma \to \mathit{enc}^{\mathit{out}}(s)\sigma$ in $\mathcal{R}(Gr)$. Since γ is a concretization and h does not contain abstract variables, we have $h\gamma = h$. Hence, we obtain $\mathit{mgu}(t\gamma, h\gamma) = \theta$ and $\gamma\theta$ is a unifier of t and h. So there must be a substitution ξ with $\sigma\xi = \gamma\theta$. Furthermore, we know that $\mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\gamma) = \varnothing$ as $\mathit{enc}^{\mathit{in}}(s)$ only contains abstract variables which must be replaced by ground terms by γ. Thus, we have $\mathit{enc}^{\mathit{in}}(s)\gamma = \mathit{enc}^{\mathit{in}}(s)\gamma\theta \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s)\gamma\theta$ by the matcher ξ and the fact that all defined function symbols in the TRSs are fresh and do not occur in the ranges of the involved substitutions, $r = 1$, and $g \geq 1$.

Let $\ell_1 > 1$. As long as we do not apply the Inst or Split rule, we can follow the concrete evaluation of $S\gamma$ through the graph with the corresponding abstract rules as we know by Thm. 5.

If we do not apply the Inst or Split rule and we do not traverse a successor of an Inst node while simulating the concrete evaluation, this simulation must end in a Suc node and, thus, we have a connection path π from s to a state $s' = (\Box_{\sigma'} \mid S'; KB')$ with $(\Box_{\theta'\theta} \mid S_{\mathit{suffix}}) = (\Box_{\sigma'} \mid S')\gamma' \in \mathcal{CON}(s')$ simulating all ℓ_1 steps of the concrete evaluation to the success state. Hence, we have the rule $\mathit{enc}^{\mathit{in}}(s)\sigma_\pi \to \mathit{enc}^{\mathit{out}}(s)\sigma_\pi$ in $\mathcal{R}(Gr)$. By Lemma 30, we have $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta)|_{\mathcal{V}(s)}$ and we obtain $\mathit{enc}^{\mathit{in}}(s)\gamma = \mathit{enc}^{\mathit{in}}(s)\gamma\theta \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s)\gamma\theta$ with the matcher γ' by the same reasoning as in the base case of the induction. Moreover, we know that the connection path has $\ell_1 + 1$ nodes. Thus, we have $r = 1$ and $g \geq \ell_1$.

If we apply the Inst or Split rule (but not to the first node) or if we reach a successor of an Inst node in more than zero steps following the concrete evaluation through the graph, then we have a connection path from s to some node s' which is no Suc node. Thus, we have the rules $\mathit{enc}^{\mathit{in}}(s)\sigma_\pi \to \mathsf{u}_{s,s'}(\mathit{enc}^{\mathit{in}}(s'), \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi))$ and $\mathsf{u}_{s,s'}(\mathit{enc}^{\mathit{out}}(s'), \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)) \to \mathit{enc}^{\mathit{out}}(s)\sigma_\pi$ in $\mathcal{R}(Gr)$. As both Inst and Split only work on states with one single goal, we know that s' only has one goal, i.e., $s' = (Q_{\sigma'\sigma_\pi}; KB')$ (where Q may additionally be labeled by a clause c). Hence, we have $S\gamma \vdash^{\ell'}_{\theta''} Q_{\theta'\theta''}\gamma' \in \mathcal{CON}(s')$ with $\ell_1 > \ell' > 0$. If s' is an Inst node, there is a node s'' in Gr which is reachable from s' by Inst edges only and $Q_{\sigma'\sigma_\pi}\gamma' \in \mathcal{CON}(s'')$. So w.l.o.g., we continue the proof for s' (take s'' for s' otherwise). Thus, s' is a start node of a connection path or a Split node. As we must have $Q_{\sigma'\sigma_\pi}\gamma' \vdash^{\ell_1-\ell'}_{\theta'''} (\Box_{\theta'\theta''\theta'''} \mid S_{\mathit{suffix}})$ for some substitution θ''' with $\theta = \theta''\theta'''$, we can use the induction hypothesis to obtain $\mathit{enc}^{\mathit{in}}(s')\gamma' \xrightarrow{\mathsf{i}}{}^{r'}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s')\gamma'\theta'''$ such that $g \cdot r' \geq \ell_1 - \ell'$. By Lemma 30, we know that $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta'')|_{\mathcal{V}(s)}$. Thus, we obtain

$$\begin{array}{rcl}
\mathit{enc}^{\mathit{in}}(s)\gamma & = & \mathit{enc}^{\mathit{in}}(s)\gamma\theta'' \\
& \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} & \mathsf{u}_{s,s'}(\mathit{enc}^{\mathit{in}}(s')\gamma', \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma') \\
& \xrightarrow{\mathsf{i}}{}^{r'}_{\mathcal{R}(Gr)} & \mathsf{u}_{s,s'}(\mathit{enc}^{\mathit{out}}(s')\gamma'\theta''', \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma') \\
& \overset{(*)}{=} & \mathsf{u}_{s,s'}(\mathit{enc}^{\mathit{out}}(s')\gamma'\theta''', \\
& & \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma'\theta''') \\
& \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} & \mathit{enc}^{\mathit{out}}(s)\sigma_\pi\gamma'\theta''' \\
& = & \mathit{enc}^{\mathit{out}}(s)\gamma\theta''\theta''' \\
& = & \mathit{enc}^{\mathit{out}}(s)\gamma\theta.
\end{array}$$

The reason for having an innermost evaluation is again the fact that the defined function symbols in the TRSs are fresh and do not occur in the ranges of the involved substitutions. To see the step $(*)$, note that $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta'')|_{\mathcal{V}(s)}$ and γ is a concretization w.r.t. $KB = (\mathcal{G}, \mathcal{U})$. In particular, it instantiates all variables in $\mathcal{G}$ by ground terms and $\mathit{enc}^{\mathit{in}}(s)$ only contains variables from $\mathcal{G}$. Thus, we have $\mathcal{V}(\mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma') = \varnothing$ and, hence, $\mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma' = \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)\sigma_\pi)\gamma'\theta'''$. Moreover, we have $r = r' + 2$ and the connection path from s to s' contains $\ell' + 1 \leq g$ nodes. Hence, we obtain $\ell_1 = \ell' + \ell_1 - \ell' \leq \ell' + g \cdot r' < g + g \cdot r' = g \cdot (r' + 1) < g \cdot (r' + 2) = g \cdot r$.

We know that s cannot be an Inst node as it must be a start node of a connection path or a Split node.

If s is a Split node, we have $S = (t, Q)_{\sigma'}$. Let $s_1 = (t_{id}; KB)$ be its first successor and $s_2 = ((Q\delta)_\delta; KB')$ its second successor. Then we have the rules $\mathit{enc}^{\mathit{in}}(s) \to \mathsf{u}_{s,s_1}(\mathit{enc}^{\mathit{in}}(s_1), \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)))$, $\mathsf{u}_{s,s_1}(\mathit{enc}^{\mathit{out}}(s_1)\delta, \mathcal{V}(\mathit{enc}^{\mathit{in}}(s))) \to \mathsf{u}_{s_1,s_2}(\mathit{enc}^{\mathit{in}}(s_2), \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)) \cup \mathcal{V}(\mathit{enc}^{\mathit{out}}(s_1)\delta))$, and $\mathsf{u}_{s_1,s_2}(\mathit{enc}^{\mathit{out}}(s_2), \mathcal{V}(\mathit{enc}^{\mathit{in}}(s)) \cup \mathcal{V}(\mathit{enc}^{\mathit{out}}(s_1)\delta)) \to \mathit{enc}^{\mathit{out}}(s)\delta$ in $\mathcal{R}(Gr)$. So we directly obtain $\mathit{enc}^{\mathit{in}}(s)\gamma \xrightarrow{\mathsf{i}}_{\mathcal{R}(Gr)} \mathsf{u}_{s,s_1}(\mathit{enc}^{\mathit{in}}(s_1)\gamma, \mathcal{V}(\mathit{enc}^{\mathit{in}}(s))\gamma)$ as the defined function symbols in the TRSs are fresh and do not occur in the range of γ. Moreover, as the concrete evaluation of $(t, Q)_{\sigma'}\gamma$ reaches a state $(\Box_{\theta'\theta} \mid S_{\mathit{suffix}})$, there must be substitutions θ'' and θ''' such that $(t\gamma)_{id} \vdash^{\ell'}_{\theta''} (\Box_{\theta''} \mid S')$ and $(Q\gamma\theta'')_{\theta''} \vdash^{\ell''}_{\theta'''} (\Box_{\theta''\theta'''} \mid S_{\mathit{suffix}})$ where $0 < \ell' < \ell_1$, $0 < \ell'' \leq \ell_1 - \ell'$, and $\theta = \theta''\theta'''$. Since s_1 must be (an instance of) a start node of a connection path and $(t\gamma)_{id}$ must also have exactly one answer substitution, we can use the induction hypothesis to obtain $\mathit{enc}^{\mathit{in}}(s_1)\gamma \xrightarrow{\mathsf{i}}{}^{r'}_{\mathcal{R}(Gr)} \mathit{enc}^{\mathit{out}}(s_1)\gamma\theta''$ with $g \cdot r' \geq \ell'$. So we have

$$\begin{array}{c}\mathsf{u}_{s,s_1}(enc^{in}(s_1)\gamma, \mathcal{V}(enc^{in}(s))\gamma) \\ \xrightarrow{\mathrm{i}}{}^{r'}_{\mathcal{R}(Gr)} \\ \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma).\end{array}$$

Furthermore, we have $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma) = \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma\theta'')$ as γ is a concretization and $enc^{in}(s)$ only contains variables from $\mathcal{G}$ for $KB = (\mathcal{G}, \mathcal{U})$. By Thm. 10 there is a concretization γ' w.r.t. KB' such that $Q\gamma\theta'' = Q\delta\gamma'$ and $t\gamma\theta'' = t\delta\gamma'$. Moreover, we know that the rule $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))) \to \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta))$ is the same as the rule $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))\delta) \to \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta)$ since $enc^{in}(s)$ only contains variables from $\mathcal{G}$ and $\mathcal{G} \cap Dom(\delta) = \varnothing$. Hence, we obtain $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta'', \mathcal{V}(enc^{in}(s))\gamma\theta'') \xrightarrow{\mathrm{i}}_{\mathcal{R}(Gr)} \mathsf{u}_{s_1,s_2}(enc^{in}(s_2)\gamma', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma')$ as the defined function symbols in the TRSs are fresh and do not occur in the ranges of the involved substitutions. As s_2 must also be (an instance of) a start node of a connection path or a SPLIT node, we can again use the induction hypothesis to obtain $enc^{in}(s_2)\gamma' \xrightarrow{\mathrm{i}}{}^{r''}_{\mathcal{R}(Gr)} enc^{out}(s_2)\gamma'\theta'''$ with $g \cdot r'' \geq \ell''$. So we have

$$\begin{array}{c}\mathsf{u}_{s_1,s_2}(enc^{in}(s_2)\gamma', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma') \\ \xrightarrow{\mathrm{i}}{}^{r''}_{\mathcal{R}(Gr)} \\ \mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma').\end{array}$$

We also know that the rule $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \to enc^{out}(s)\delta$ is the same as the rule $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta) \to enc^{out}(s)\delta$ since, again, δ does not instantiate any variables in $enc^{in}(s)$. Moreover,

$$\begin{array}{c}(\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma' \\ = \\ (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma'\theta''',\end{array}$$

because we have $\delta\gamma' = \gamma\theta''$ on all variables in $enc^{in}(s)$ and $enc^{out}(s_1)$ and by the groundness analysis we are guaranteed to instantiate all these variables by ground terms using $\gamma\theta''$. This yields $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma') = \mathsf{u}_{s_1,s_2}(enc^{out}(s_2)\gamma'\theta''', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)))\delta\gamma'\theta''') \xrightarrow{\mathrm{i}}_{\mathcal{R}(Gr)} enc^{out}(s)\delta\gamma'\theta''' = enc^{out}(s)\gamma\theta''\theta''' = enc^{out}(s)\gamma\theta$ since, again, the defined function symbols in the TRSs are fresh and do not occur in the ranges of the involved substitutions. Altogether, we have $enc^{in}(s)\gamma \xrightarrow{\mathrm{i}}{}^{r'+r''+3}_{\mathcal{R}(Gr)} enc^{out}(s)\gamma\theta$, i.e., $r = r' + r'' + 3$. It remains to show that $g \cdot r \geq \ell_1$.

We know that $\ell_1 = \ell' + \ell''$ since the evaluation of $(\square_{\theta'\theta} \mid S_{suffix} \mid S')$ belongs to the ℓ_2 trailing evaluation steps. So we obtain $\ell_1 = \ell' + \ell'' \leq g \cdot r' + g \cdot r'' = g \cdot (r' + r'') < g \cdot (r' + r'' + 3) = g \cdot r$. □

LEMMA 33 (COMPLEXITY APPROXIMATION). *Let Gr be a decomposable symbolic evaluation graph and let $S\gamma$ be a concretization of an abstract state $s = (S; KB)$ in Gr which is a start node of a connection path or a* SPLIT *node. Moreover, let Gr_s be the subgraph of Gr at s and let the first ℓ steps of the longest evaluation of $S\gamma$ be simulated within Gr_s (i.e., the end of this prefix of the concrete evaluation is reached as soon as we reach a multiplicative* SPLIT *node during the simulation — if we do not reach such a node, the complete concrete evaluation can be simulated within Gr_s). Furthermore, let $g_s = |Gr_s|$ denote the number of nodes in Gr_s and $g_s^{\text{SPLIT}} = |Split(Gr_s) \setminus mults(Gr)|$ denote the number of non-multiplicative* SPLIT *nodes in Gr_s Then we have $\ell < g_s$ or there is an innermost evaluation w.r.t. $\mathcal{R}(Gr)$ starting from $enc^{in}(s)\gamma$ with r rewrite steps using a rule from $\mathcal{R}(Gr_s)$ such that $2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r \geq \ell$.*

PROOF. We prove the lemma by induction on ℓ. If $\ell = 0$, we trivially have $\ell < g_s$. If $\ell = 1$, then we either have $\ell < g_s$ or the only node in Gr_s is s, i.e., it is a multiplicative SPLIT node. But then the prefix of the evaluation cannot have the length $\ell = 1$ as its simulation starts at a multiplicative SPLIT node and, thus, must stop directly.

For $\ell > 1$ we can assume that the proposition holds for all evaluations with a prefix of a length which is at most $\ell - 1$ which can be simulated within Gr_s. If $\ell < g_s$, there is nothing to show. So let $\ell \geq g_s$. We perform a case analysis on s.

If s is a SPLIT node, then we know that s is not multiplicative (since we have $\ell > 1$) and S contains a single goal resulting from a query (t, Q). Let $s_1 = ((t)_{id}; KB)$ be the first successor of s and $s_2 = ((Q\delta)_\delta; KB')$ be the second successor of s. Furthermore, we have the rules $enc^{in}(s) \to \mathsf{u}_{s,s_1}(enc^{in}(s_1), \mathcal{V}(enc^{in}(s)))$, $\mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))) \to \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta))$, and $\mathsf{u}_{s_1,s_2}(enc^{out}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \to enc^{out}(s)\delta$ in $\mathcal{R}(Gr)$ and $\mathcal{R}(Gr_s)$.

If there is no answer substitution for $t\gamma$ computed during the prefix, then we know that the prefix which can be simulated within Gr_s of the evaluation of $S\gamma$ is the same as the prefix which can be simulated within Gr_s of the evaluation of $t\gamma$. Moreover, we know that s_1 must be (an instance of) a start node of a connection path. So we have $enc^{in}(s)\gamma \xrightarrow{\mathrm{i}}_{\mathcal{R}(Gr_s)} C[enc^{in}(s_1)\gamma]$ and we can continue the proof in the case where s is a start node of a connection path as s_1 is such a node and the prefix of the evaluation of $t\gamma \in \mathcal{CON}(s_1)$ has the same length as the prefix of the evaluation of $S\gamma$. Note that the rewrite step is innermost as the function symbols introduced in the TRSs are fresh and do not occur in the range of γ.

If there are answer substitutions for $t\gamma$ computed during the prefix, then we know that exactly one answer substitution θ_1 is computed for $t\gamma$ since s is not multiplicative. Thus, the prefix of the evaluation of $S\gamma$ has the form $(t\gamma, Q\gamma)_\theta \vdash^{\ell_1'} ((Q\gamma\theta_1)_{\theta\theta_1} \mid S_1) \vdash^{\ell_Q} S_1 \vdash^{\ell_f} S_{end}$ where the evaluation from the state $((Q\gamma\theta_1)_{\theta\theta_1} \mid S_1)$ to the state S_1 corresponds to the longest evaluation of $(Q\gamma\theta_1)_{\theta\theta_1}$ or has the form $(t\gamma, Q\gamma)_\theta \vdash^{\ell_1'} ((Q\gamma\theta_1)_{\theta\theta_1} \mid S_1) \vdash^{\ell_Q} S_{end}$ where the evaluation from the state $((Q\gamma\theta_1)_{\theta\theta_1} \mid S_1)$ to the state S_{end} corresponds to the prefix of the longest evaluation of $(Q\gamma\theta_1)_{\theta\theta_1}$. We directly see that the length of the prefix of the evaluation of $t\gamma$ is $\ell_t = 1 + \ell_f + \ell_1'$ (where $\ell_f = 0$ if it does not exist in the prefix) as this is the sum of the lengths of evaluation parts not belonging to the evaluation of $Q\gamma\theta_1$ plus one application of the SUC rule to add the answer substitution instead of continuing the evaluation with $Q\gamma\theta_1$. We also know that $0 < \ell_t < \ell$ and $0 < \ell_Q < \ell$. Since s is not multiplicative, we have $\ell_f < g_s$. Furthermore, we again have $enc^{in}(s)\gamma \xrightarrow{\mathrm{i}}_{\mathcal{R}(Gr_s)} \mathsf{u}_{s,s_1}(enc^{in}(s_1)\gamma, \mathcal{V}(enc^{in}(s))\gamma)$. By the fact that we actually obtain the answer substitution during the prefix of the evaluation, we know that its simulation does not traverse a multiplicative SPLIT node. Thus, we obtain $enc^{in}(s_1)\gamma \xrightarrow{\mathrm{i}}{}^{r'}_{\mathcal{R}(Gr_{s_1})} enc^{out}(s_1)\gamma\theta_1$ by Lemma 32. Moreover, by Thm. 10 there must be a concretization γ' w.r.t.

KB' such that $\gamma\theta_j = \delta\gamma'$ on all variables in s, s_1, and s_2. We also know that δ does not instantiate variables in $enc^{in}(s)$ and, thus, the rule

$$\begin{gathered} \mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))) \\ \to \\ \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)) \end{gathered}$$

is the same as the rule

$$\begin{gathered} \mathsf{u}_{s,s_1}(enc^{out}(s_1)\delta, \mathcal{V}(enc^{in}(s))\delta) \\ \to \\ \mathsf{u}_{s_1,s_2}(enc^{in}(s_2), \mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta)). \end{gathered}$$

So we have

$$\begin{gathered} \mathsf{u}_{s,s_1}(enc^{in}(s_1)\gamma, \mathcal{V}(enc^{in}(s))\gamma) \\ \overset{\mathsf{i}}{\to}{}^{r'}_{\mathcal{R}(Gr_{s_1})} \\ \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta_1, \mathcal{V}(enc^{in}(s))\gamma) \\ = \\ \mathsf{u}_{s,s_1}(enc^{out}(s_1)\gamma\theta_1, \mathcal{V}(enc^{in}(s))\gamma\theta_1) \\ \overset{\mathsf{i}}{\to}_{\mathcal{R}(Gr_s)} \\ \mathsf{u}_{s_1,s_2}(enc^{in}(s_2)\gamma', (\mathcal{V}(enc^{in}(s)) \cup \mathcal{V}(enc^{out}(s_1)\delta))\gamma') \end{gathered}$$

as the defined function symbols in the TRSs are fresh and do not occur in the ranges of the involved substitutions. Since s_2 must be (an instance of) a SPLIT node or a start node of a connection path, we can use the induction hypothesis to obtain that $\ell_Q < g_{s_2}$ or to obtain an innermost evaluation with r'' rewrite steps using rules from $\mathcal{R}(Gr_{s_2})$ starting from $enc^{in}(s_2)\gamma'$ with $2 \cdot g_{s_2} \cdot 2^{g_{s_2}^{\text{SPLIT}}} \cdot r'' \geq \ell_Q$. We obviously have $g_{s_1} \leq g_s$, $g_{s_2} \leq g_s$, $g_{s_1}^{\text{SPLIT}} \leq g_s^{\text{SPLIT}}$, $g_{s_2}^{\text{SPLIT}} \leq g_s^{\text{SPLIT}}$, $\mathcal{R}(Gr_{s_1}) \subseteq \mathcal{R}(Gr_s)$, and $\mathcal{R}(Gr_{s_2}) \subseteq \mathcal{R}(Gr_s)$ as both s_1 and s_2 are reachable from s without traversing multiplicative SPLIT nodes. If $\ell_Q < g_{s_2}$, we have a derivation with $r = 2 + r'$ rewrite steps using rules from $\mathcal{R}(Gr_s)$ starting from $enc^{in}(s)\gamma$. So we obtain $\ell \overset{\ell_Q<g_s}{<} \ell_t + g_s \overset{\ell_f<g_s}{<} \ell'_1 + 1 + g_s + g_s \leq g_{s_1} \cdot r' + 2 \cdot g_s + 1 \leq g_s \cdot r' + 2 \cdot g_s + 1 \overset{g_s>1, g_s^{\text{SPLIT}}\geq 1}{<} 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r' + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot 2 = 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r$. If $\ell_Q \geq g_{s_2}$, then we append the derivation with r'' rewrite steps using rules from $\mathcal{R}(Gr_{s_2})$ to the derivation from $enc^{in}(s)\gamma$ to $C[enc^{in}(s_2)\gamma']$. So we obtain an innermost derivation with $r = 2 + r' + r''$ rewrite steps using rules from $\mathcal{R}(Gr_s)$. Thus, we have $\ell = \ell_t + \ell_Q \overset{\ell_f<g_s}{<} \ell'_1 + \ell_Q + g_s + 1 \leq g_{s_1} \cdot r' + 2 \cdot g_{s_2} \cdot 2^{g_{s_2}^{\text{SPLIT}}} \cdot r'' + g_s + 1 \leq g_s \cdot r' + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r'' + g_s + 1 \overset{g_s>1, g_s^{\text{SPLIT}}\geq 1}{<} 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r' + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r'' + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot 2 = 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot (r' + r'' + 2) = 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r$.

If s is a start node of a connection path, the beginning of $S\gamma$'s derivation corresponds to the traversal of a connection path in Gr by Thm. 5. Note that we can follow this connection path until we reach an INST or SPLIT node or the successor of an INST node, and we let $s' = (S'; KB')$ be the end node of this connection path π (i.e., we do not stop at SUC nodes). By Thm. 5, there is an $S'\gamma' \in \mathcal{CON}(s')$ with $S\gamma \vdash^{\ell'} S'\gamma'$ and the prefix of the derivation of $S'\gamma'$ has the length $\ell - \ell'$. Moreover, $\mathcal{R}(Gr_s)$ (and thus, $\mathcal{R}(Gr)$) contains the rule $enc^{in}(s)\sigma_\pi \to \mathsf{u}(enc^{in}(s'), \mathcal{V}(enc^{in}(s)\sigma_\pi))$ for some fresh function symbol u. By Def. 7 and 9, we know that s only contains one single goal. Thus, by Lemma 30 we obtain $(\sigma_\pi\gamma')|_{\mathcal{V}(s)} = (\gamma\theta)|_{\mathcal{V}(s)}$. Furthermore, we have $\mathcal{V}(enc^{in}(s)\gamma) = \varnothing$ and, thus, $enc^{in}(s)\gamma = enc^{in}(s)\gamma\theta = enc^{in}(s)\sigma_\pi\gamma' \overset{\mathsf{i}}{\to}_{\mathcal{R}(Gr_s)} C[enc^{in}(s')\gamma']$. Again, this is an innermost rewrite step since the function symbols f_s^{in} and u introduced in the TRSs are fresh and do not occur in the ranges of σ_π or γ'. Moreover, s' is (an instance of) a SPLIT node or a start node of a connection path. Thus, we can use the induction hypothesis to obtain that $\ell - \ell' < g_{s'}$ or to obtain an innermost derivation with r' rewrite steps using rules from $\mathcal{R}(Gr_{s'})$ and starting from $enc^{in}(s')\gamma'$ such that $2 \cdot g_{s'} \cdot 2^{g_{s'}^{\text{SPLIT}}} \cdot r' \geq \ell - \ell'$. We obviously have $g_{s'} \leq g_s$, $g_{s'}^{\text{SPLIT}} \leq g_s^{\text{SPLIT}}$, and $\mathcal{R}(Gr_{s'}) \subseteq \mathcal{R}(Gr_s)$ as s' is reachable from s without traversing multiplicative SPLIT nodes. If $\ell - \ell' < g_{s'}$, then we have an innermost derivation with $r = 1$ rewrite steps using rules from $\mathcal{R}(Gr_s)$ and starting from $S\gamma$. Moreover, we know that $\ell' < g_s$ as π contains $\ell' + 1 \leq g_s$ nodes. So we have $\ell = \ell' + \ell - \ell' < g_s + g_{s'} \leq 2 \cdot g_s < 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r$, since $g_s > 1$, $g_s^{\text{SPLIT}} \geq 0$, and $r = 1$. If $\ell - \ell' \geq g_{s'}$, then we append the derivation with r' rewrite steps using rules from $\mathcal{R}(Gr_{s'})$ to the derivation of length 1 from $enc^{in}(s)\gamma$ to $C[enc^{in}(s')\gamma']$ and obtain an innermost derivation with $r = r' + 1$ rewrite steps using rules from $\mathcal{R}(Gr_s)$. We still know that $\ell' < g_s$ and, thus, we have $\ell = \ell' + \ell - \ell' < g_s + \ell - \ell' \leq g_s + 2 \cdot g_{s'} \cdot 2^{g_{s'}^{\text{SPLIT}}} \cdot r' \leq g_s + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r' \leq 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r'$, since $g_s^{\text{SPLIT}} \geq 0$. Moreover, $2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} + 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r' = 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot (r' + 1) = 2 \cdot g_s \cdot 2^{g_s^{\text{SPLIT}}} \cdot r$. □

Now we can finally prove Thm. 23. The proof only needs to be performed for finite evaluations since Thm. 17 already implies Thm. 23 for infinite evaluations. Thus, we can use induction on the length of the evaluation (similar to the proof of Thm. 16). The over-approximation of the asymptotic runtime is ensured by the fact that the length of each connection path is bounded by the number of nodes in the evaluation graph. Hence, while one rewrite step simulates several evaluation steps, the number of simulated evaluation steps by one rewrite step is bounded by a constant. Since the SPLIT nodes are not multiplicative, their first successors only produce at most one answer substitution. Thus, the runtimes for simulating evaluations from the successors of such SPLIT nodes just have to be added. This is ensured by the construction of the *SplitRules*, which call the rules corresponding to the two successors of the SPLIT node after each other.

THEOREM 23 (SOUNDNESS OF COMPLEXITY ANALYSIS I). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$. If Gr has no multiplicative* SPLIT *nodes then $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(irc_{\mathcal{R}(Gr)}(n))$.*

PROOF. We already know by Thm. 17 that if $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) = \infty$ (i.e., the logic program does not terminate for the specified query set), then the resulting TRS has an infinite innermost rewrite sequence. By the proof of Thm. 17, the sequence starts with a term of the form $enc^{in}(s)\gamma$, which is basic. Thus, we also have $irc_{\mathcal{R}(Gr)}(n) = \infty$. So we only need to show the theorem for logic programs which are terminating w.r.t. the specified sets of queries.

Obviously, Lemma 31 and Lemma 33 imply the theorem for finite derivations as the root node must be (an instance of) a start node of a connection path if there is an evaluation whose length exceeds the number of nodes in Gr. Moreover, if all evaluations have a length smaller than the number of nodes in Gr, we have $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(1) \subseteq \mathcal{O}(irc_{\mathcal{R}}(n))$ for all TRSs $\mathcal{R}$. Furthermore, g_s, g_s^{SPLIT}, and k_s^{max} as defined in Lemma 33 do not depend on n in $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n)$ or $irc_{\mathcal{R}(Gr)}(n)$, as for each SPLIT node s' and for each $S\gamma \in Succ_1(s')$, the number of answer substitutions for $S\gamma$ is bounded by a constant number. Finally, if a graph as in Lemma 33 has no multiplicative SPLIT nodes, then we have $Gr_{root(Gr)} = Gr$ and the considered prefix of the evaluation covers the whole evaluation. $\square$

The following definition is needed in the proof of Thm. 27.

DEFINITION 34 (RELATIVE EVALUATION LENGTH). *Let $\mathcal{R}$ be a TRS, $\mathcal{R}' \subseteq \mathcal{R}$, and $d = (t \to_{\ell_1 \to r_1} t_1 \to_{\ell_2 \to r_2} \dots \to_{\ell_k \to r_k} t_k)$ be an evaluation w.r.t. $\mathcal{R}$. Then the* relative length *$RelLength_{\mathcal{R}'}(d)$ is $|\{i \mid \ell_i \to r_i \in \mathcal{R}'\}|$.*

The proof of Thm. 27 relies on Thm. 23 which already implies Thm. 27 for evaluation graphs without multiplicative SPLIT nodes. Evaluations within a subgraph (i.e., which do not traverse multiplicative SPLIT nodes) can be simulated analogously. As soon as an evaluation reaches a multiplicative SPLIT node, the remaining evaluation is at most as long as the length of the evaluation of the first successor of that SPLIT node plus the number of answer substitutions for this first successor times the length of the evaluation of the second successor for one (the "worst") answer substitution of the first successor. As the number of answer substitutions for the first successor is bounded by the runtime of the first successor, we obtain that the overall runtime is bounded by the multiplication of the runtimes for the first and the second successor (the latter covers all possible answer substitutions and, thus, also the "worst" one).

THEOREM 27 (SOUNDNESS OF COMPLEXITY ANALYSIS II). *Let $\mathcal{P}$ be a logic program, $\mathsf{p} \in \Sigma$, m a moding function, and let Gr be a symbolic evaluation graph for $\mathcal{P}$ whose root is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$. If Gr is decomposable, then we have $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) \in \mathcal{O}(cplx_{root(Gr)}(n))$.*

PROOF. Again, for non-terminating evaluations, this is implied by Thm. 17. So we only need to consider finite evaluations.

Let $root(Gr) = (S; KB)$. By Lemma 31, we know that $enc^{in}(root(Gr))\gamma$ is a basic term w.r.t. $\mathcal{R}(Gr)$ for all concretizations γ w.r.t. KB.

We extend the definition of the prc function to work for arbitrary abstract states (so not only for the root state of a symbolic evaluation graph resp. not only for the corresponding class of queries). To this end, we say that for an abstract state s, $prc_{\mathcal{P},\mathcal{CON}(s)}$ maps any $n \in \mathbb{N}$ to the length of the longest evaluation starting with a concrete state $S\gamma \in \mathcal{CON}(s)$ with $|enc^{in}(s)\gamma| \le n$. By Lemma 31, we know that this is in fact an extension of the previous definition of $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}$, since we have $prc_{\mathcal{P},\mathcal{Q}_m^{\mathsf{p}}}(n) = prc_{\mathcal{P},\mathcal{CON}(root(Gr))}(n)$ for all $n \in \mathbb{N}$.

We now show the following proposition (*) for finite concrete evaluations: Let s be a SPLIT node or a start node of a connection path in a decomposable symbolic evaluation graph Gr. Then we have $prc_{\mathcal{P},\mathcal{CON}(s)}(n) \in \mathcal{O}(cplx_s(n))$.

Obviously, proposition (*) implies the theorem as $root(Gr)$ is (an instance of) a start node of a connection path and we know $\mathcal{Q}_m^{\mathsf{p}} = \mathcal{CON}(root(Gr))$ by Lemma 31.

We show proposition (*) by induction on the number g^{mult} of multiplicative SPLIT nodes in Gr. If $g^{mult} = 0$, then we have $cplx_s(n) = irc_{\mathcal{R}(Gr_s),\mathcal{R}(Gr)}(n) = irc_{\mathcal{R}(Gr)}(n)$ and, thus, the proposition holds by Thm. 23. For $g^{mult} > 0$, we can assume that the proposition holds for all decomposable graphs with at most $g^{mult} - 1$ multiplicative SPLIT nodes. Let $S\gamma \in \mathcal{CON}(s)$ and let the longest evaluation of $S\gamma$ have the length ℓ.

If s is a multiplicative SPLIT node, we have $S\gamma = (t\gamma, Q\gamma)_\theta$. Let $Succ_1(s) = s_1$ and $Succ_2(s) = s_2$. Then we have $cplx_s(n) = cplx_{Succ_1(s)}(n) \cdot cplx_{Succ_2(s)}(n)$ for all $n \in \mathbb{N}$. Let there be k answer substitutions for $t\gamma$ and the longest evaluation of $t\gamma$ have the length ℓ_t. Moreover, let the longest evaluation of $Q\gamma\theta'$ for any answer substitution θ' of $t\gamma$ have the length ℓ_Q. Then we have $\ell \le \ell_t + k \cdot \ell_Q$. Since we have for all $s \in mults(Gr)$ that both s_1 and s_2 cannot reach s in Gr, we know that both Gr_{s_1} and Gr_{s_2} have less multiplicative SPLIT nodes than Gr. Thus, we can use the induction hypothesis to obtain $prc_{\mathcal{P},\mathcal{CON}(Succ_1(s))}(n) \in \mathcal{O}(cplx_{Succ_1(s)}(n))$ and $prc_{\mathcal{P},\mathcal{CON}(Succ_2(s))}(n) \in \mathcal{O}(cplx_{Succ_2(s)}(n))$ for all $n \in \mathbb{N}$. If $\ell_t > k \cdot \ell_Q$, then we have $\ell < 2 \cdot \ell_t$ and we obtain $prc_{\mathcal{P},\mathcal{CON}(s)}(n) \in \mathcal{O}(cplx_{Succ_1(s)}(n)) \subseteq \mathcal{O}(cplx_{Succ_1(s)}(n) \cdot cplx_{Succ_2(s)}(n)) = \mathcal{O}(cplx_s(n))$. If $\ell_t \le k \cdot \ell_Q$, we have $\ell \le 2 \cdot k \cdot \ell_Q$. As we cannot have more answer substitutions than evaluation steps, we know that $k \le \ell_t$. Thus, we have $\ell \le 2 \cdot \ell_t \cdot \ell_Q$ and obtain $prc_{\mathcal{P},\mathcal{CON}(s)}(n) \in \mathcal{O}(cplx_{Succ_1(s)}(n) \cdot cplx_{Succ_2(s)}(n)) = \mathcal{O}(cplx_s(n))$.

If s is no multiplicative SPLIT node, we have $cplx_s(n) = irc_{\mathcal{R}(Gr_s),\mathcal{R}(Gr)}(n) + \Sigma_{s' \in mults(Gr) \cap Gr_s} cplx_{s'}(n)$. If the simulation of the longest concrete evaluation of $S\gamma$ does not traverse a multiplicative SPLIT node, the proposition follows by Lemma 33. So we consider the case that the simulation of the longest concrete evaluation of $S\gamma$ does traverse a multiplicative SPLIT node. Let s' be the first such node where the evaluation of $S\gamma$ reaches a state $S'\gamma' \in \mathcal{CON}(s')$ with ℓ' steps. By Lemma 33, we obtain $\ell' \le c \cdot RelLength_{\mathcal{R}(Gr_s)}(d)$ for a constant c. By an analogous reasoning as for the case where s is a multiplicative SPLIT node, we furthermore obtain $prc_{\mathcal{P},\mathcal{CON}(s')}(n) \in \mathcal{O}(cplx_{s'}(n))$. So together we obtain $prc_{\mathcal{P},\mathcal{CON}(s)}(n) \in \mathcal{O}(irc_{\mathcal{R}(Gr_s),\mathcal{R}(Gr)}(n) + cplx_{s'}(n)) = \mathcal{O}(cplx_s(n))$. $\square$

The following lemma is needed for the proof of Thm. 28.

LEMMA 35 (NO SUCCESS WITHOUT SUC NODES). *Let Gr be a symbolic evaluation graph. Let s be a node in Gr which does not reach any* SUC *node in Gr. Then for any concretization of s, its evaluation does not produce any answer substitutions, i.e., the* SUC *rule is never applied in the evaluation.*

PROOF. Let γ be a concretization w.r.t. KB where $s = (S; KB)$. We perform the proof of the lemma by contradiction, assuming that from $S\gamma$ in ℓ steps we first reach a node where the SUC rule is applied. We use induction over the lexicographic combination of the length ℓ of the evaluation of $S\gamma$ and of the edge relation of Gr restricted to INST and SPLIT edges. Note that this induction relation is indeed well founded as the number of terms in a state is strictly decreased when applying the SPLIT rule while it stays equal when applying the INST rule. So every infinite sequence of rule applications with only these rules must eventually only use the INST rule. This is in contradiction to the requirement that symbolic evaluation graphs do not contain cycles consisting of INST edges only.

For $\ell = 0$, we trivially have that the number of answer substitutions produced for $S\gamma$ is 0.

For $\ell > 0$, we perform a case analysis on s:

- If $s \in \mathit{Inst}(Gr)$, then we know by Thm. 8 that $S\gamma \in \mathcal{CON}(\mathit{Succ}_1(s))$. So we can directly use the induction hypothesis to obtain the lemma.
- If $s \in \mathit{Split}(Gr)$, then we have $S = (t, Q)_\sigma$ and we know that the evaluation of $t\gamma$ is at most as long as the one of $S\gamma$. Thus, we can use the induction hypothesis on the first successor of s (there we have one less SPLIT edge to traverse) and obtain that no answer substitution is produced for $t\gamma$. It follows immediately that then $S\gamma$ can also produce no answer substitutions (since Q is only called if $t\gamma$ succeeds, which is not the case).
- Otherwise, we applied a rule to s where the next concrete state of the evaluation is represented by one successor of s by Thm. 5. We obtain the lemma by the induction hypothesis.

□

The proof of Thm. 28 can be performed by contradiction. We use induction on the length of a prefix of an evaluation reaching two SUC states.

THEOREM 28 (SOUNDNESS OF DETERMINACY CRITERION). *Let $\mathcal{P}$ be a logic program and let Gr be a symbolic evaluation graph for $\mathcal{P}$. Let s be a node in Gr such that for all* SUC *nodes s' reachable from s, there is no non-empty path from s' to a* SUC *node. Then s is deterministic. Thus, if s is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$ for a $\mathsf{p} \in \Sigma$ and a moding function m, then all queries in $\mathcal{Q}_m^{\mathsf{p}}$ are also deterministic.*

PROOF. Let γ be a concretization w.r.t. KB where $s = (S; KB)$. We perform the proof of the theorem by contradiction, assuming that from $S\gamma$ in ℓ steps for the second time we reach a node where the SUC rule is applied. We use induction over the lexicographic combination of the length ℓ of the evaluation of $S\gamma$ and of the edge relation of Gr restricted to INST and SPLIT edges. Note that this induction relation is indeed well founded as the number of terms in a state is strictly decreased when applying the SPLIT rule while it stays equal when applying the INST rule. So every infinite sequence of rule applications with only these rules must eventually only use the INST rule. This is in contradiction to the requirement that symbolic evaluation graphs do not contain cycles consisting of INST edges only.

For $\ell = 0$, we trivially have that the number of answer substitutions produced for $S\gamma$ is $0 \leq 1$.

For $\ell > 0$, we perform a case analysis on s:

- If $s \in \mathit{Inst}(Gr)$, then we know by Thm. 8 that $S\gamma \in \mathcal{CON}(\mathit{Succ}_1(s))$. So we can use the induction hypothesis directly to obtain the theorem.
- If $s \in \mathit{Suc}(Gr)$, then we know that the evaluation must start with one application of the SUC rule resulting in a concrete state S' and producing one answer substitution for this step. Moreover, we know that $S' \in \mathcal{CON}(\mathit{Succ}_1(s))$. Since we know that $\mathit{Succ}_1(s)$ cannot reach any SUC node, we obtain by Lemma 35 that the evaluation of S' does not produce any additional answer substitutions and, hence, it follows that the evaluation of $S\gamma$ produces exactly one answer substitution.
- If $s \in \mathit{Split}(Gr)$, then we have $S = (t, Q)_\sigma$ and we know that the evaluation of $t\gamma$ is at most as long as the one of $S\gamma$. Thus, we can use the induction hypothesis on the first successor of s and obtain that at most one answer substitution δ is produced for $t\gamma$. Hence, in case the evaluation reaches a state $Q\gamma\delta$, this state is represented by $\mathit{Succ}_2(s)$ as we know by Thm. 10. The induction hypothesis is therefore applicable to $\mathit{Succ}_2(s)$ as well and we obtain that the evaluation of $Q\gamma\delta$ also produces at most one answer substitution. Together, the evaluation of $S\gamma$ produces at most one answer substitution as the numbers of solutions for $t\gamma$ and $Q\gamma\delta$ have to be multiplied. If already $t\gamma$ does not produce any solutions, so does $S\gamma$ and the theorem holds.
- Otherwise, we applied a rule to s where the next concrete state of the evaluation is represented by one successor of s (as we know by Thm. 5) and no answer substitution is produced in that step (i.e., the SUC rule is not applied). We obtain the theorem by the induction hypothesis.

□

We can improve the determinacy criterion a little bit further by exploiting that a SPLIT node cannot be successful (and hence, is deterministic) if at least one of its successors is not successful (even if the other successor is not deterministic). So suppose that on any path from s to a SUC node, one traverses a SPLIT node $r = ((t, Q); KB)$. If one of the sub-queries t or Q always fails (i.e., one of r's successors has no path to a SUC node), then the composed query (t, Q) fails as well. Thus, there is also no answer substitution for s and consequently, s is deterministic. However, this improvement is only useful in rare situations (in our experiments, we gained just one example by this improvement). The improved determinacy criterion is stated below.

DEFINITION 36 (IMPROVED DETERMINACY CRITERION). *For any node s in Gr let $\mathit{Reach}_{Gr}(s)$ consist of all nodes of Gr that can be reached from s. In particular, we always have $s \in \mathit{Reach}_{Gr}(s)$. A node s in Gr satisfies the* improved determinacy criterion *if each* SUC *node $s' \in \mathit{Reach}_{Gr}(s)$ satisfies condition (a) or (b):*

(a) Gr has no non-empty path from s' to a SUC *node.*

(b) On every path from s to s', there is a SPLIT *node r where $\mathit{Reach}_{Gr}(\mathit{Succ}_1(r))$ or $\mathit{Reach}_{Gr}(\mathit{Succ}_2(r))$ contains no* SUC *node.*

The next and final theorem shows that this improved criterion also implies determinacy of all queries represented by the root of a symbolic evaluation graph.

THEOREM 37 (SOUNDNESS OF DETERMINACY ANALYSIS). *Let $\mathcal{P}$ be a logic program, let Gr be a symbolic evaluation graph for $\mathcal{P}$ and let s be a node in Gr which satisfies the improved determinacy criterion in Def. 36. Then s is deterministic. Thus, if s is the initial state corresponding to $\mathcal{Q}_m^{\mathsf{p}}$ for a $\mathsf{p} \in \Sigma$ and a moding function m, then all queries in $\mathcal{Q}_m^{\mathsf{p}}$ are also deterministic.*

PROOF. The proof is analogous to the proof of Thm. 28. We only need to show that as soon as a successor of a SPLIT node cannot reach a SUC node, the SPLIT node must be deterministic. So let $s = ((t, Q)_\sigma; KB)$ be a SPLIT node and let γ be a concretization w.r.t. KB. If one of $\mathit{Succ}_1(s)$ or

$Succ_2(s)$ cannot reach any SUC node, we know by Lemma 35 that each concretization of this successor does not produce any answer substitutions. Since $(t, Q)\gamma$ produces as many answer substitutions as the product of the number of answer substitutions produced by $t\gamma$ and $Q\gamma\theta$ for each answer substitution θ of $t\gamma$, we obtain that $(t, Q)\gamma$ produces no answer substitutions as $t\gamma$ or all $Q\gamma\theta$ cannot produce any answer substitutions. Thus, the theorem holds. □

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2009-01 *	Fachgruppe Informatik: Jahresbericht 2009
2009-02	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications
2009-03	Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices
2009-05	George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs
2009-06	George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete
2009-07	Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I
2009-08	Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs
2009-09	Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem
2009-10	Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm
2009-11	Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs
2009-12	Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes
2009-13	Martin Zimmermann: Time-optimal Winning Strategies for Poset Games
2009-14	Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)
2009-15	Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems
2009-16	George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs
2009-17	Carsten Kern: Learning Communicating and Nondeterministic Automata
2009-18	Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-01 *	Fachgruppe Informatik: Jahresbericht 2010
2010-02	Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time
2010-03	Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering
2010-04	René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme
2010-05	Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme
2010-06	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07	George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08	Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09	George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10	Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11	Martin Zimmermann: Parametric LTL Games
2010-12	Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13	Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-14	Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination
2010-15	Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode
2010-16	Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles
2010-17	Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten
2010-18	Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit
2010-19	Felix Reidl: Experimental Evaluation of an Independent Set Algorithm
2010-20	Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games
2011-01 *	Fachgruppe Informatik: Jahresbericht 2011
2011-02	Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting
2011-03	Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems
2011-04	Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars
2011-06	Johannes Lotz, Klaus Leppkes, and Uwe Naumann: dco/c++ - Derivative Code by Overloading in C++

2011-07	Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV
2011-08	Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog
2011-09	Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing
2011-10	Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations
2011-11	Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains
2011-12	Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems
2011-13	Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP
2011-14	Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games
2011-16	Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM
2011-18	Kamal Barakat: Introducing Timers to pi-Calculus
2011-19	Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode
2011-24	Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations
2011-25	Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations
2011-26	Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata
2012-01 *	Fachgruppe Informatik: Annual Report 2012
2012-02	Thomas Heer: Controlling Development Processes
2012-03	Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems
2012-04	Marcus Gelderie: Strategy Machines and their Complexity
2012-05	Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting
2012-06	Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data
2012-07	André Egners, Björn Marschollek, and Ulrike Meyer : Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms
2012-08	Hongfei Fu: Computing Game Metrics on Markov Decision Processes
2012-09	Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer : Quantitative Timed Analysis of Interactive Markov Chains