Scheduling Complexity of Interleaving Search

Dmitry Rozplokhas[0000−0001−7882−4497] and
Dmitry Boulytchev[0000−0001−8363−7143]

St Petersburg University and JetBrains Research, Russia
rozplokhas@gmail.com, dboulytchev@math.spbu.ru

Abstract. miniKanren is a lightweight embedded language for logic and relational programming. Many of its useful features come from a distinctive search strategy, called *interleaving search*. However, with interleaving search conventional ways of reasoning about the complexity and performance of logical programs become irrelevant. We identify an important key component — *scheduling* — which makes the reasoning for miniKanren so different, and present a semi-automatic technique to estimate the scheduling impact via symbolic execution for a reasonably wide class of programs.

Keywords: miniKanren, interleaving search, time complexity, symbolic execution

1 Introduction

A family of embedded languages for logic and, more specifically, relational programming miniKanren [10] has demonstrated an interesting potential in various fields of program synthesis and declarative programming [5, 6, 14]. A distinctive feature of miniKanren is *interleaving search* [13] which, in particular, delivers such an important feature as completeness.

However, being a different search strategy than conventional BFS/DFS/iterative deepening, etc., interleaving search makes the conventional ways of reasoning about the complexity of logical programs not applicable. Moreover, some intrinsic properties of interleaving search can manifest themselves in a number of astounding and, at the first glance, unexplainable performance effects.

As an example, let's consider two implementations of list concatenation relation (Fig. 1, left side); we respect here a conventional tradition for miniKanren programming to superscript all relational names with "o". The only difference between the two is the position of the recursive call. The evaluation of these implementations on the same problem (Fig. 1, right side) shows that the first implementation works significantly slower, although it performs exactly the same number of unifications. As a matter of fact, these two implementations even have different *asymptotic* complexity under the assumption that occurs check is disabled.[1] Although the better performance of the $\texttt{append}^o_{opt}$ relation is expected even under conventional strategies due to tail recursion, the asymptotic difference is striking.

[1] The role of occurs check is discussed in Section 5.

```
append^o =λ a b ab .
  ((a ≡ Nil) ∧ (ab ≡ b)) ∨
  (fresh (h t tb)
     (a ≡ Cons(h, t)) ∧
     (append^o t b tb) ∧
     (ab ≡ Cons(h, tb)))

append^o_opt =λ a b ab .
  ((a ≡ Nil) ∧ (ab ≡ b)) ∨
  (fresh (h t tb)
     (a ≡ Cons(h, t)) ∧
     (ab ≡ Cons(h, tb) ∧
     (append^o_opt t b tb)))
```

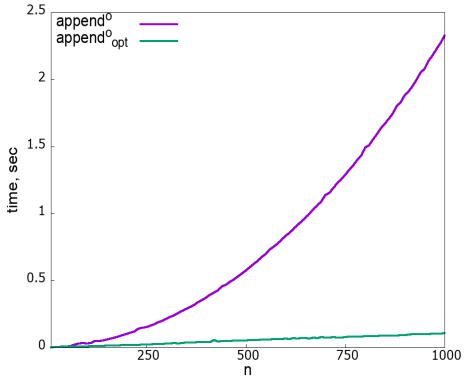

Fig. 1. Two implementations of list concatenation and their performance for $a = [1, \ldots, n]$, $b = [1, \ldots, 100]$, and ab left free.

A careful analysis discovers that the difference is caused not by unifications, but by the process of *scheduling* goals during the search. In miniKanren a lazy structure is maintained to decompose the goals into unifications, perform these unifications in a certain order, and thread the results appropriately. For the $\texttt{append}^o_{opt}$ relation the size of this structure is constant, while for the $\texttt{append}^o$ this structure becomes linear in size, reducing the performance.

This paper presents a formal framework for scheduling cost complexity analysis for interleaving search in miniKanren. We use the reference operational semantics, reflecting the behaviour of actual implementations [15], and prove the soundness of our approach w.r.t. this semantics. The roadmap of the approach is as follows: we identify two complexity measures (one of which captures *scheduling complexity*) and give exact and approximate recursive formulae to calculate them (Section 3); then we present a procedure to automatically extract inequalities for the measures for a given goal using symbolic execution (Section 4). These inequalities have to be reformulated and solved manually in terms of a certain *metatheory*, which, on success, provides asymptotic bounds for the scheduling complexity of a goal evaluation. Our approach puts a number of restrictions on the goal being analyzed as well as on the relational program as a whole. We explicitly state these restrictions in Section 2 and discuss their impact in Section 7. The proofs of all lemmas and theorems can be found in the extended version of this paper. [2]

2 Background: Syntax and Semantics of miniKanren

In this section, we recollect some known formal descriptions for miniKanren language that will be used as a basis for our development. The descriptions here are taken from [15] (with a few non-essential adjustments for presentation purposes) to make the paper self-contained, more details and explanations can be found there.

[2] https://arxiv.org/abs/2202.08511

$$
\begin{array}{lll}
\mathcal{C} & = \{C_i^{k_i}\} & \text{constructors} \\
\mathcal{T}_X & = X \cup \{C_i^{k_i}(t_1, \ldots, t_{k_i}) \mid t_j \in \mathcal{T}_X\} & \text{terms over the set of variables } X \\
\mathcal{D} & = \mathcal{T}_\varnothing & \text{ground terms} \\
\mathcal{X} & = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots\} & \text{syntactic variables} \\
\mathcal{A} & = \{x, y, z \ldots\} & \text{logic variables} \\
\mathcal{R} & = \{R_i^{k_i}\} & \text{relational symbols with arities} \\
\\
\mathcal{G} & = \mathcal{T}_\mathcal{X} \equiv \mathcal{T}_\mathcal{X} & \text{equality} \\
 & \quad \mathcal{G} \wedge \mathcal{G} & \text{conjunction} \\
 & \quad \mathcal{G} \vee \mathcal{G} & \text{disjunction} \\
 & \quad \textbf{fresh } \mathcal{X} \,.\, \mathcal{G} & \text{fresh variable introduction} \\
 & \quad R_i^{k_i}(t_1, \ldots, t_{k_i}),\ t_j \in \mathcal{T}_\mathcal{X} & \text{relational symbol invocation} \\
\\
\mathcal{S} & = \{R_i^{k_i} = \lambda\ \mathtt{x}_1^i \ldots \mathtt{x}_{k_i}^i \,.\, g_i;\}\ g,\ g_i, g \in \mathcal{G} & \text{specification}
\end{array}
$$

Fig. 2. The syntax of miniKanren

$$
\begin{array}{lll}
\Sigma = \mathcal{A} \to \mathcal{T}_\mathcal{A} & \text{substitutions} & S = \langle \mathcal{G},\ E \rangle \text{ task} \\
E = \Sigma \times \mathbb{N} & \text{environments} & S \oplus S \text{ sum} \\
 & & S \otimes \mathcal{G} \text{ product} \\
L = \circ \mid E & \text{labels} & \hat{S} = \diamond \mid S \text{ states}
\end{array}
$$

Fig. 3. States and labels in the LTS for miniKanren

The syntax of core miniKanren is shown in Fig. 2. All data is presented using terms $\mathcal{T}_X$ built from a fixed set of constructors $\mathcal{C}$ with known arities and variables from a given set X. We parameterize the terms with an alphabet of variables since in the semantic description we will need *two* kinds of variables: *syntactic* variables $\mathcal{X}$, used for bindings in the definitions, and *logic* variables $\mathcal{A}$, which are introduced and unified during the evaluation. We assume the set $\mathcal{A}$ is ordered and use the notation α_i to specify a position of a logical variable w.r.t. this order.

There are five types of goals: unification of two terms, conjunction and disjunction of goals, fresh logic variable introduction, and invocation of some relational definition. For the sake of brevity, in code snippets, we abbreviate immediately nested "**fresh**" constructs into the one, writing "**fresh** `x y` $\ldots$ g" instead of "**fresh** `x` . **fresh** `y` . $\ldots$ g". The *specification* $\mathcal{S}$ consists of a set of relational definitions and a top-level goal. A top-level goal represents a search procedure that returns a stream of substitutions for the free variables of the goal.

During the evaluation of miniKanren program an environment, consisting of a substitution for logic variables and a counter of allocated logic variables, is threaded through the computation and updated in every unification and fresh variable introduction. The substitution in the environment at a given point and given branch of evaluation contains all the information about relations between the logical variables at this point. Different branches are combined via *interleaving search* procedure [13]. The answers for a given goal are extracted from the final environments.

$$\langle t_1 \equiv t_2,\, (\sigma, n)\rangle \xrightarrow{\circ} \Diamond,\ \nexists\, mgu\,(t_1\sigma, t_2\sigma) \qquad [\textsc{UnifyFail}]$$

$$\langle t_1 \equiv t_2,\, (\sigma, n)\rangle \xrightarrow{(mgu\,(t_1\sigma, t_2\sigma)\circ\sigma),\, n)} \Diamond \qquad [\textsc{UnifySuccess}]$$

$$\langle\, \mathbf{fresh}\ \mathtt{x}\,.\,g,\, (\sigma, n)\rangle \xrightarrow{\circ} \langle g\,[\alpha_{n+1}\,/_{\mathtt{x}}],\, (\sigma, n+1)\rangle \qquad [\textsc{Fresh}]$$

$$\frac{R_i^{k_i} = \lambda\, \mathtt{x}_1 \ldots \mathtt{x}_{k_i}\,.\,g}{\langle R_i^{k_i}\,(t_1, \ldots, t_{k_i}),\, e\rangle \xrightarrow{\circ} \langle g\,[t_1\,/_{\mathtt{x}_1}\ldots t_{k_i}\,/_{\mathtt{x}_{k_i}}],\, e\rangle} \qquad [\textsc{Invoke}]$$

$$\langle g_1 \vee g_2,\, e\rangle \xrightarrow{\circ} \langle g_1,\, e\rangle \oplus \langle g_2,\, e\rangle \qquad [\textsc{Disj}] \qquad \langle g_1 \wedge g_2,\, e\rangle \xrightarrow{\circ} \langle g_1,\, e\rangle \otimes g_2 \qquad [\textsc{Conj}]$$

$$\frac{s_1 \xrightarrow{l} \Diamond}{(s_1 \oplus s_2) \xrightarrow{l} s_2} \qquad [\textsc{DisjStop}] \qquad \frac{s_1 \xrightarrow{l} s_1'}{(s_1 \oplus s_2) \xrightarrow{l} (s_2 \oplus s_1')} \qquad [\textsc{DisjStep}]$$

$$\frac{s \xrightarrow{\circ} \Diamond}{(s \otimes g) \xrightarrow{\circ} \Diamond} \qquad [\textsc{ConjStop}] \qquad \frac{s \xrightarrow{e} \Diamond}{(s \otimes g) \xrightarrow{\circ} \langle g,\, e\rangle} \qquad [\textsc{ConjStopAns}]$$

$$\frac{s \xrightarrow{\circ} s'}{(s \otimes g) \xrightarrow{\circ} (s' \otimes g)} \qquad [\textsc{ConjStep}] \qquad \frac{s \xrightarrow{e} s'}{(s \otimes g) \xrightarrow{\circ} (\langle g,\, e\rangle \oplus (s' \otimes g))} \qquad [\textsc{ConjStepAns}]$$

Fig. 4. Operational semantics of interleaving search

This search procedure is formally described by operational semantics in the form of a labeled transition system. This semantics corresponds to the canonical implementation of interleaving search.

The form of states and labels in the transition system is defined in Fig. 3. Non-terminal states S have a tree-like structure with intermediate nodes corresponding to partially evaluated conjunctions ("$\otimes$") or disjunctions ("$\oplus$"). A leaf in the form $\langle g,\, e\rangle$ determines a task to evaluate a goal g in an environment e. For a conjunction node, its right child is always a goal since it cannot be evaluated unless some result is provided by the left conjunct. We also need a terminal state $\diamond$ to represent the end of the evaluation. The label "$\circ$" is used to mark those steps which do not provide an answer; otherwise, a transition is labeled by an updated environment.

The transition rules are shown in Fig. 4. The first six rules define the evaluation of leaf states. For the disjunction and conjunction, the corresponding node states are constructed. For other types of goals the environment and the evaluated goal are updated in accordance with the task given by the goal: for an equality the most general unifier of the terms is incorporated into the substitution (or execution halts if the terms are non-unifiable); for a fresh construction a new variable is introduced and the counter of allocated variables is incremented; for a relational call the body of the relation is taken as the next goal. The rest of the rules define composition of evaluation of substates for partial disjunctions and conjunctions. For a partial disjunction, the first constituent is evaluated for one step, then the constituents are swapped (which constitutes the *interleaving*), and the label is propagated. When the evaluation of the first constituent of partial disjunction halts, the evaluation proceeds with the second constituent. For a partial conjunction, the first constituent is evaluated until the answer is obtained, then the evaluation of the second constituent with this answer as the environment is scheduled for evaluation together with the remaining partial conjunction (via partial disjunction node). When the evaluation of the first

$$
\begin{array}{l}
B_{nf} = \mathcal{T}_{\mathcal{X}} \equiv \mathcal{T}_{\mathcal{X}} \;\mid\; R^k\,(\mathcal{T}_{\mathcal{X}}, \ldots, \mathcal{T}_{\mathcal{X}}) \\
C_{nf} = B_{nf} \;\mid\; C_{nf} \wedge B_{nf} \\
F_{nf} = C_{nf} \;\mid\; \textbf{fresh}\ X\ .\ F_{nf} \\
D_{nf} = F_{nf} \;\mid\; D_{nf} \vee F_{nf}
\end{array}
$$

Fig. 5. Disjunctive Normal Form for goals

constituent of partial conjunction halts, the evaluation of the conjunction halts, too.

The introduced transition system is completely deterministic, therefore a derivation sequence for a state s determines a certain *trace* — a sequence of states and labeled transitions between them. It may be either finite (ending with the terminal state $\Diamond$) or infinite. We will denote by $\mathcal{T}r^{st}(s)$ the sequence of states in the trace for initial state s and by $\mathcal{T}r^{ans}(s)$ the sequence of answers in the trace for initial state s. The sequence $\mathcal{T}r^{ans}(s)$ corresponds to the stream of answers in the reference miniKanren implementations.

In the following we rely on the following property of leaf states:

Definition 2.1. *A leaf state $\langle g, (\sigma, n)\rangle$ is well-formed iff $\mathcal{FV}\,(g) \cup \mathcal{D}om\,(\sigma) \cup \mathcal{VR}an\,(\sigma) \subseteq \{\alpha_1, \ldots, \alpha_n\}$, where $\mathcal{FV}\,(g)$ denotes the set of free variables in a goal g, $\mathcal{D}om\,(\sigma)$ and $\mathcal{VR}an\,(\sigma)$ — the domain of a substitution σ and a set of all free variables in its image respectively.*

Informally, in a well-formed leaf state all free variables in goals and substitution respect the counter of free logical variables. This definition is in fact an instance of a more general definition of well-formedness for all states, introduced in [15], where it is proven that the initial state is well-formed and any transition from a well-formed state results in a well-formed one.

Besides operational semantics, we will make use of a denotational one analogous to the least Herbrand model. For a relation R^k, its denotational semantics $[\![R^k]\!]$ is treated as a k-ary relation on the set of all ground terms, where each "dimension" corresponds to a certain argument of R^k. For example, $[\![\texttt{append}^o]\!]$ is a set of all triplets of ground lists, in which the third component is a concatenation of the first two. The concrete description of the denotational semantics is given in [15] as well as the proof of the soundness and completeness of the operational semantics w.r.t. to the denotational one.

Finally, we explicitly enumerate all the restrictions required by our method to work:

- All relations have to be in DNF (set D_{nf} in Fig.5).
- We only consider goals which converge with a finite number of answers.
- All answers have to be ground (*groundness* condition) for all relation invocations encountered during the evaluation.
- All answers have to be unique (*answer uniqueness* condition) for all relation invocations encountered during the evaluation.

3 Scheduling Complexity

We may notice that the operational semantics described in the previous section can be used to calculate the exact number of elementary scheduling steps. Our

first idea is to take the number of states $d\,(s)$ in the finite trace for a given state s:

$$d\,(s) \stackrel{\text{def}}{=} |\mathcal{T}r^{st}(s)|$$

However, it turns out that this value alone does not provide an accurate scheduling complexity estimation. The reason is that some elementary steps in the semantics are not elementary in existing implementations. Namely, a careful analysis discovers that each semantic step involves navigation to the leftmost leaf of the state which in implementations corresponds to multiple elementary actions, whose number is proportional to the height of the leftmost branch of the state in question. Here we provide an *ad-hoc* definition for this value, $t\,(s)$, which we call the *scheduling factor*:

$$t\,(s) \stackrel{\text{def}}{=} \sum_{s_i \in \mathcal{T}r^{st}(s)} lh\,(s_i)$$

where $lh\,(s_i)$ is the height of the leftmost branch of the state.

In the rest of the section, we derive recurrent equations which would relate the scheduling complexity for states to the scheduling complexity for their (immediate) substates. It turns out that to come up with such equations both t and d values have to be estimated simultaneously.

The next lemma provides the equations for $\oplus$-states:

Lemma 3.1. *For any two states s_1 and s_2*

$$d\,(s_1 \oplus s_2) = d\,(s_1) + d\,(s_2)$$
$$t\,(s_1 \oplus s_2) = t\,(s_1) + t\,(s_2) + cost_{\oplus}(s_1 \oplus s_2)$$

where $cost_{\oplus}(s_1 \oplus s_2) = \min\{2 \cdot d\,(s_1) - 1, 2 \cdot d\,(s_2)\}$

Informally, for a state in the form $s_1 \oplus s_2$ the substates are evaluated separately, one step at a time for each substate, so the total number of semantic steps is the sum of those for the substates. However, for the scheduling factor, there is an extra summand $cost_{\oplus}(s_1 \oplus s_2)$ since the "leftmost heights" of the states in the trace are one node greater than those for the original substates due to the introduction of one additional $\oplus$-node on the top. This additional node persists in the trace until the evaluation of one of the substates comes to an end, so the scheduling factor is increased by the number of steps until that.

The next lemma provides the equations for $\otimes$-states:[3]

Lemma 3.2. *For any state s and any goal g*

$$d\,(s \otimes g) = d\,(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\,(\langle g,\, a_i\rangle) \qquad (\star)$$

$$t\,(s \otimes g) = t\,(s) + cost_{\otimes}(s \otimes g) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} (t\,(\langle g,\, a_i\rangle) + cost_{\oplus}(\langle g,\, a_i\rangle \oplus (s'_i \otimes g))) \qquad (\dagger)$$

where

$$cost_{\otimes}(s \otimes g) = d\,(s)$$
$$s'_i = \textit{the first state in the trace for } s \textit{ after a transition delivering the answer } a_i$$

[3] We assume $\Diamond \otimes g = \Diamond$

For the states of the form $s \otimes g$ the reasoning is the same, but the resulting equations are more complicated. In an $\otimes$-state the left substate is evaluated until an answer is found, which is then taken as *an environment* for the evaluation of the right subgoal. Thus, in the equations for $\otimes$-states the evaluation times of the second goal *for all the answers* generated for the first substate are summed up. The evaluation of the right subgoal in different environments is added to the evaluation of the left substate via creation of an $\oplus$-state, so for the scheduling factor there is an additional summand $cost_{\oplus}(\langle g,\, a_i\rangle \oplus s'_i)$ for each answer with s'_i being the state after discovering the answer. There is also an extra summand $cost_{\otimes}(s \otimes g)$ for the scheduling factor because of the $\otimes$-node that increases the height in the trace, analogous to the one caused by $\oplus$-nodes. Note, a $\otimes$-node is always placed immediately over the left substate so this addition is exactly the number of steps for the left substate.

Unfolding costs definitions in (†) gives us a cumbersome formula that includes some intermediate states s'_i encountered during the evaluation. However, as ultimately we are interested in asymptotic estimations, we can approximate these costs up to a multiplicative constant. We can notice that the value $d\,(s'_i \otimes g)$ occurring in the second argument of $cost_{\oplus}$ includes values $d\,(\langle g,\, a_j\rangle)$ (like in the first argument) for all answers a_j after this intermediate state. So in the sum of all $cost_{\oplus}$ values $d\,(\langle g,\, a_i\rangle)$ may be excluded for at most one answer, and in fact, if we take the maximal one of these values we will get a rather precise approximation. Specifically, we can state the following approximation[4] for $t\,(s \otimes g)$.

Lemma 3.3.

$$t\,(s \otimes g) = t\,(s) + \left(\sum_{a_i \in \mathcal{T}r^{ans}(s)} t\,(\langle g,\, a_i\rangle)\right) + \Theta\left(d\,(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\,(\langle g,\, a_i\rangle) - \dot{\max}_{a_i \in \mathcal{T}r^{ans}(s)} d\,(\langle g,\, a_i\rangle)\right)$$

Hereafter we use the following notation: $\dot{\max} S = \max\,(S \cup \{0\})$. We can see that the part under Θ is very similar to the $(\star)$ except that here we exclude d value for one of the answers from the sum. This difference is essential and, as we will see later, it is in fact responsible for the difference in complexities for our motivating example.

4 Complexity Analysis via Symbolic Execution

Our approach to complexity analysis is based on a semi-automatic procedure involving symbolic execution. In the previous section, we presented formulae to compositionally estimate the complexity factors for *non-leaf states* of operational semantics under the assumption that corresponding estimations for *leaf states* are given. In order to obtain corresponding estimations for relations as a whole, we would need to take into account the effects of relational invocations, including the recursive ones.

[4] We assume the following definition for $f\,(x) = g\,(x) + \Theta\,(h\,(x))$:

$$\exists C_1, C_2 \in \mathcal{R}^+,\, \forall x : g\,(x) + C_1 \cdot h\,(x) \leq f\,(x) \leq g\,(x) + C_2 \cdot h\,(x)$$

Another observation is that as a rule we are interested in complexity estimations in terms of some *metatheory*. For example, dealing with relations on lists we would be interested in estimations in terms of list lengths, with trees — in terms of depth or number of nodes, with numbers — in terms of their values, etc. It is unlikely that a generic term-based framework would provide such specific information automatically. Thus, a viable approach would be to extract some inequalities involving the complexity factors of certain relational calls automatically and then let a human being solve these inequalities in terms of a relevant metatheory.

For the sake of clarity we will provide a demonstration of complexity analysis for a specific example — $\texttt{append}^o$ relation from the introduction — throughout the section.

The extraction procedure utilizes a symbolic execution technique and is completely automatic. It turns out that the semantics we have is already abstract enough to be used for symbolic execution with minor adjustments. In this symbolic procedure, we mark some of the logic variables as "grounded" and at certain moments substitute them with ground terms. Informally, for some goal with some free logic variables we consider the complexity of a search procedure which finds the bindings for all non-grounded variables based on the ground values substituted for the grounded ones. This search procedure is defined precisely by the operational semantics; however, as the concrete values of grounded variables are unknown (only the fact of their *groundness*), the whole procedure becomes symbolic. In particular, in unification the groundness can be propagated to some non-grounded free variables. Thus, the symbolic execution is determined by a set of grounded variables (hereafter denoted as $V \subset \mathcal{A}$). The initial choice of V determines the problem we analyze.

In our example the objective is to study the execution when we specialize the first two arguments with ground values and leave the last argument free. Thus, we start with the goal $\texttt{append}^o\ a\ b\ ab$ (where a, b and ab are distinct free logic variables) and set the initial $V = \{a, b\}$.

We can make an important observation that both complexity factors (d and t) are stable w.r.t. the renaming of free variables; moreover, they are also stable w.r.t. the change of the fresh variables counter as long as it stays adequate, and change of current substitution, as long as it gives the same terms after application. Formally, the following lemma holds.

Lemma 4.1. *Let* $s = \langle g, (\sigma, n)\rangle$ *and* $s' = \langle g', (\sigma', n')\rangle$ *be two well-formed states. If there exists a bijective substitution* $\pi\colon FV(g\sigma) \to FV(g'\sigma')$ *such that* $g\sigma\pi = g'\sigma'$, *then* $d(s) = d(s')$ *and* $t(s) = t(s')$.

The lemma shows that the set of states for which a call to relation has to be analyzed can be narrowed down to a certain family of states.

Definition 4.1. *Let* g *be a goal. An initial state for* g *is* $init(g) = \langle g, (\varepsilon, n_{init}(g))\rangle$ *with* $n_{init}(g) = \min\{n \mid FV(g) \subseteq \{\alpha_1 \dots \alpha_n\}\}$

Due to the Lemma 4.1 it is sufficient for analysis of a relational call to consider only the family of initial states since an arbitrary call state encountered

throughout the execution can be transformed into an initial one while preserving both complexity factors. Thus, the analysis can be performed in a compositional manner where each call can be analyzed separately. For our example the family of initial states is $q^{app}(\mathbf{a}, \mathbf{b}) = init\,(\texttt{append}^o\ \mathbf{a}\ \mathbf{b}\ ab)$ for arbitrary ground terms $\mathbf{a}$ and $\mathbf{b}$.

As we are aiming at the complexity estimation depending on specific ground values substituted for grounded variables, in general case extracted inequalities have to be parameterized by *valuations* — mappings from the set of grounded variables to ground terms. As the new variables are added to this set during the execution, the valuations need to be extended for these new variables. The following definition introduces this notion.

Definition 4.2. *Let $V \subset U \subset \mathcal{A}$ and $\rho\colon V \to \mathcal{T}_\varnothing$ and $\rho'\colon U \to \mathcal{T}_\varnothing$ be two valuations. We say that ρ' extends ρ (denotation: $\rho' \succ \rho$) if $\rho'(x) = \rho(x)$ for all $x \in V$.*

The main objective of the symbolic execution in our case is to find constraints on valuations for every leaf goal in the body of a relation that determine whether the execution will continue and how a valuation changes after this goal. For internal relational calls, we describe constraints in terms of denotational semantics (to be given some meaning in terms of metatheory later). We can do it because of a precise equivalence between the answers found by operational semantics and values described by denotational semantics thanks to soundness and completeness as well as our requirements of grounding and uniqueness of answers. Our symbolic treatment of equalities relies on the fact that substitutions of ground terms commute, in a certain sense, with unifications. More specifically, we can use the most general unifier for two terms to see how unification goes for two terms with some free variables substituted with ground terms. The most general unifier may contain bindings for both grounded and non-grounded variables. A potential most general unifier for terms after substitution contains the same bindings for non-grounding terms (with valuation applied to their rhs), while bindings for grounding variables turn into equations that should be satisfied by the unifier with ground value on the left and bound term on the right. In particular, this means that all variables in bindings for grounded variables become grounded, too. We can use this observation to define an iterative process that determines the updated set of grounded variables $\mathbf{upd}\,(U, \delta)$ for a current set U and a most general unifier δ and a set of equations $\mathbf{constr}\,(\delta, U)$ that should be respected by the valuation.

$$\mathbf{upd}\,(U, \delta) = \begin{cases} U & \forall x \in U : FV\,(\delta\,(x)) \subset U \\ \mathbf{upd}\,(U \cup \bigcup\limits_{x \in U} FV\,(\delta\,(x)),\ \delta) & \text{otherwise} \end{cases}$$

$$\mathbf{constr}\,(\delta, U) = \{x = \delta\,(x) \mid x \in U \cap \mathcal{D}om\,(\delta)\}$$

Using these definitions we can describe symbolic unification by the following lemma.

Lemma 4.2. *Let t_1, t_2 be terms, $V \subset \mathcal{A}$ and $\rho\colon V \to \mathcal{T}_\varnothing$ be a valuation. If $mgu\,(t_1, t_2) = \delta$ and $U = \boldsymbol{upd}\,(V, \delta)$ then $t_1\rho$ and $t_2\rho$ are unifiable iff there is*

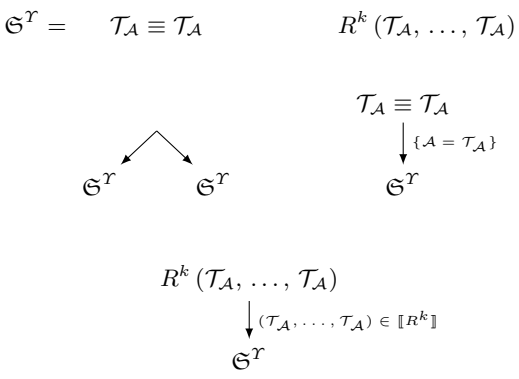

Fig. 6. Symbolic Scheme Forms

some $\rho' \colon U \to \mathcal{T}_{\varnothing}$ *such that* $\rho' \succ \rho$ *and* $\forall (y = t) \in \boldsymbol{constr}(\delta,\, U) \,:\, \rho'(y) = t\rho'$. *In such case* ρ' *is unique and* $\rho \circ mgu\,(t_1\rho, t_2\rho) = \delta \circ \rho'$ *up to alpha-equivalence (e.g. there exists a bijective substitution* $\pi : FV(t_1) \to FV(t_2)$, *s.t.* $\rho \circ mgu\,(t_1\rho, t_2\rho) = \delta \circ \rho' \circ \pi$*).*

In our description of the extraction process, we use a visual representation of symbolic execution of a relation body for a given set of grounded variables in a form of a *symbolic scheme*. A symbolic scheme is a tree-like structure with different branches corresponding to execution of different disjuncts and nodes corresponding to equalities and relational calls in the body augmented with subsets of grounded variables at the point of execution.[5] Constraints for substituted grounded variables that determine whether the execution continues are presented as labels on the edges of a scheme.

Each scheme is built as a composition of the five patterns, shown in Fig. 6 (all schemes are indexed by subsets of grounded variables with $\Upsilon = 2^{\mathcal{A}}$ denoting such subsets).

Note, the constraints after nodes of different types differ: unification puts a constraint in a form of a set of equations on substituted ground values that should be respected while relational call puts a constraint in a form of a tuple of ground terms that should belong to the denotational semantics of a relation.

The construction of a scheme for a given goal (initially, the body of a relation) mimics a regular execution of a relational program. The derivation rules for scheme formation have the following form $\langle \Gamma,\, \sigma,\, n,\, V \rangle \;\vdash\; g \;\rightsquigarrow\; \mathfrak{S}^V$. Here g is a goal, Γ is a list of *deferred* goals (these goals have to be executed after the execution of g in every branch in the same order, initially this list is empty; this resembles continuations, but the analogy is not complete), σ and n are substitution and counter from the current environment respectively, V is a set of grounded variables at the moment.

[5] Note the difference with conventional symbolic execution graphs with different branches representing mutually exclusive paths of evaluation, not the different parts within one evaluation.

$$\frac{\langle g_2 : \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g_1 \;\rightsquigarrow\; \mathfrak{S}^V}{\langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g_1 \wedge g_2 \;\rightsquigarrow\; \mathfrak{S}^V} \quad [\textsc{Conj}_{\mathfrak{S}}]$$

$$\frac{\langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g_1 \;\rightsquigarrow\; \mathfrak{S}_1{}^V,\; \langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g_2 \;\rightsquigarrow\; \mathfrak{S}_2{}^V}{\langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g_1 \vee g_2 \;\rightsquigarrow\; \begin{array}{cc} \mathfrak{S}_1{}^V & \mathfrak{S}_2{}^V \end{array}} \quad [\textsc{Disj}_{\mathfrak{S}}]$$

$$\frac{\langle \Gamma,\, \sigma,\, n+1,\, V\rangle \;\vdash\; g\,[\alpha_n\,/_{\mathtt{x}}] \;\rightsquigarrow\; \mathfrak{S}^V}{\langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; \mathbf{fresh}\ \mathtt{x}\ .\ g \;\rightsquigarrow\; \mathfrak{S}^V} \quad [\textsc{Fresh}_{\mathfrak{S}}]$$

$$\langle \epsilon,\, \sigma,\, n,\, V\rangle \;\vdash\; t_1 \equiv t_2 \;\rightsquigarrow\; t_1\sigma \equiv t_2\sigma \quad [\textsc{UnifyLeaf}_{\mathfrak{S}}]$$

$$\langle \epsilon,\, \sigma,\, n,\, V\rangle \;\vdash\; R^k\,(t_1,\, \ldots,\, t_k) \;\rightsquigarrow\; R^k\,(t_1\sigma,\, \ldots,\, t_k\sigma) \quad [\textsc{InvokeLeaf}_{\mathfrak{S}}]$$

$$\frac{\nexists mgu\,(t_1\sigma, t_2\sigma)}{\langle g : \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; t_1 \equiv t_2 \;\rightsquigarrow\; t_1\sigma \equiv t_2\sigma} \quad [\textsc{UnifyFail}_{\mathfrak{S}}]$$

$$\frac{mgu\,(t_1\sigma, t_2\sigma) = \delta,\; U = \mathbf{upd}\,(V,\, \delta),\; \langle \Gamma,\, \sigma\delta,\, n,\, U\rangle \;\vdash\; g \;\rightsquigarrow\; \mathfrak{S}^U}{\langle g : \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; t_1 \equiv t_2 \;\rightsquigarrow\; \begin{array}{c} t_1\sigma \equiv t_2\sigma \\ \big\downarrow \mathbf{constr}\,(\delta,\; U) \\ \mathfrak{S}^U \end{array}} \quad [\textsc{UnifySuccess}_{\mathfrak{S}}]$$

$$\frac{U = V \cup \bigcup_i FV\,(t_i\sigma),\; \langle \Gamma,\, \sigma,\, n,\, U\rangle \;\vdash\; g \;\rightsquigarrow\; \mathfrak{S}^U}{\langle g : \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; R^k\,(t_1,\, \ldots,\, t_k) \;\rightsquigarrow\; \begin{array}{c} R^k\,(t_1\sigma,\, \ldots,\, t_k\sigma) \\ \big\downarrow (t_1\sigma,\, \ldots,\, t_k\sigma) \in [\![R^k]\!] \\ \mathfrak{S}^U \end{array}} \quad [\textsc{Invoke}_{\mathfrak{S}}]$$

Fig. 7. Scheme Formation Rules

The rules are shown in Fig. 7. [$\textsc{Conj}_{\mathfrak{S}}$] and [$\textsc{Disj}_{\mathfrak{S}}$] are structural rules: when investigating conjunctions we defer the second conjunct by adding it to Γ and continue with the first conjunct; disjunctions simply result in forks. [$\textsc{Fresh}_{\mathfrak{S}}$] introduces a fresh logic variable (not grounded) and updates the counter of occupied variables. When the investigated goal is equality or relational call it is added as a node to the scheme. If there are no deferred goals, then this node is a leaf (rules [$\textsc{UnifyLeaf}_{\mathfrak{S}}$] and [$\textsc{InvokeLeaf}_{\mathfrak{S}}$]). Equality is also added as a leaf if there are some deferred goals, but the terms are non-unifiable and so the execution stops (rule [$\textsc{UnifyFail}_{\mathfrak{S}}$]). If the terms in the equality are unifiable and there are deferred goals (rule [$\textsc{UnifySuccess}_{\mathfrak{S}}$]), the equality is added as a node and the execution continues for the deferred goals, starting from the leftmost one; also the set of grounded variables is updated and constraint labels are added for the edge in accordance with Lem. 4.2. For relational calls the

proccess is similar: if there are some deferred goals (rule [INVOKE$_{\mathfrak{S}}$]), all variables occurring in a call become grounded (due to the grounding condition we imposed) and should satisfy the denotational semantics of the invoked relation.

The scheme constructed by these rules for our `append`^{o} example is shown in Fig. 8. For simplicity, we do not show the set of grounded variables for each node, but instead overline grounded variables in-place. Note, all variables that occur in constraints on the edges are grounded after parent node execution.

Now, we can use schemes to see how the information for leaf goals in relation body is combined with conjunctions and disjunctions. Then we can apply formulae from Section 3 to get recursive inequalities (providing lower and upper bounds simultaneously) for both complexity factors.

In these inequalities we need to sum up the values of d and t-factor for all leaf goals of a body and for all environments these goals are evaluated for. The leaf goals are the nodes of the scheme and evaluated environments can be derived from the constraints attached to the edges. So, for this summation we introduce the following notions: $\mathcal{D}$ is the sum of d-factor values and $\mathcal{T}$ is the sum of t-factor values for the execution of the body with specific valuation ρ.

Their definitions are shown in Fig. 9 (both formulas are given in the same figure as the definitions coincide modulo factor denotations). For nodes, we take the corresponding value (for equality it always equals 1). When going through an equality we sum up the rest with an updated valuation (by Lem. 4.2 this sum always has one or zero summands depending on whether the unification succeeds or not). When going through a relational call we take a sum of all valuations that satisfy the denotational semantics (these valuations will correspond exactly to the set of all answers produced by the call since operational semantics is sound and complete w.r.t. the denotational one and because we require all answers to be unique). For disjunctions, we take the sum of both branches.

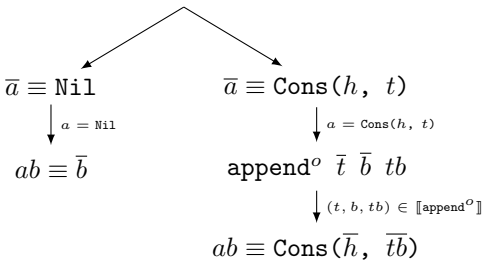

Fig. 8. Symbolic execution scheme for the goal `append`^{o} a b ab with initial set of grounded variables $V = \{a, b\}$. For each node, variables that are grounded at the point of execution of this node are overlined.

As we saw in Section 3 when computing the scheduling factors we need to exclude from the additional cost the value of d-factor for one of the environments (the largest one). This is true for the generalized formula for a whole scheme, too. This time we need to take all executed environments for all the leaves of a scheme and exclude the d-factor value for a maximal one (the formula for conjunction ensures that we make the exclusion for the leaf, and the formula for disjunction ensures that we make it for only one of the leaves). So, we will need additional notion $\mathcal{L}$, similar to $\mathcal{D}$ and $\mathcal{T}$ that will collect all the goals of the form

$$
\begin{array}{lll}
{}^{D}\!/_{\mathcal{T}}\,(& t_2 \equiv t_2 &)(\rho) = 1 \\
{}^{D}\!/_{\mathcal{T}}\,(& R^k\,(t_1,\,\ldots,\,t_k) &)(\rho) = {}^{d}\!/_{t}\,(init\,(R^k\,(t_1\rho,\,\ldots,\,t_k\rho))) \\
{}^{D}\!/_{\mathcal{T}}\,(& \begin{array}{c} t_1 \equiv t_2 \\ \downarrow {\scriptstyle Cs} \\ \mathfrak{S}^U \end{array} &)(\rho) = 1 + \sum\limits_{\substack{\rho' :\, V \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ \forall (y,t) \in Cs :\, \rho'(y) =_t \rho'}} {}^{D}\!/_{\mathcal{T}}\,(\mathfrak{S}^U)(\rho') \\
{}^{D}\!/_{\mathcal{T}}\,(& \begin{array}{c} R^k\,(t_1,\,\ldots,\,t_k) \\ \downarrow {\scriptstyle (t_1,\,\ldots,\,t_k)\,\in\,[\![R^k]\!]} \\ \mathfrak{S}^U \end{array} &)(\rho) = {}^{d}\!/_{t}\,(init\,(R^k\,(t_1\rho,\,\ldots,\,t_k\rho))) + \sum\limits_{\substack{\rho' :\, V \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ (t_1\rho',\ldots,t_k\rho') \in [\![R^k]\!]}} {}^{D}\!/_{\mathcal{T}}\,(\mathfrak{S}^U)(\rho') \\
{}^{D}\!/_{\mathcal{T}}\,(& \begin{array}{c} \swarrow \quad \searrow \\ {\mathfrak{S}_1}^V \qquad {\mathfrak{S}_2}^V \end{array} &)(\rho) = {}^{D}\!/_{\mathcal{T}}\,({\mathfrak{S}_1}^V)(\rho) + {}^{D}\!/_{\mathcal{T}}\,({\mathfrak{S}_2}^V)(\rho)
\end{array}
$$

Fig. 9. Complexity Factors Extraction: $\mathcal{D}$ and $\mathcal{T}$

$$
\begin{array}{lll}
\mathcal{L}\,(& t_2 \equiv t_2 &)(\rho) = \{init\,(t_2 \equiv t_2)\} \\
\mathcal{L}\,(& R^k\,(t_1,\,\ldots,\,t_k) &)(\rho) = \{init\,(R^k\,(t_1\rho,\,\ldots,\,t_k\rho))\} \\
\mathcal{L}\,(& \begin{array}{c} t_1 \equiv t_2 \\ \downarrow {\scriptstyle Cs} \\ \mathfrak{S}^U \end{array} &)(\rho) = \bigcup\limits_{\substack{\rho' :\, V \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ \forall (y,t) \in Cs :\, \rho'(y) =_t \rho'}} \mathcal{L}\,(\mathfrak{S}^U)(\rho') \\
\mathcal{L}\,(& \begin{array}{c} R^k\,(t_1,\,\ldots,\,t_k) \\ \downarrow {\scriptstyle (t_1,\,\ldots,\,t_k)\,\in\,[\![R^k]\!]} \\ \mathfrak{S}^U \end{array} &)(\rho) = \bigcup\limits_{\substack{\rho' :\, V \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ (t_1\rho',\ldots,t_k\rho') \in [\![R^k]\!]}} \mathcal{L}\,(\mathfrak{S}^U)(\rho') \\
\mathcal{L}\,(& \begin{array}{c} \swarrow \quad \searrow \\ {\mathfrak{S}_1}^V \qquad {\mathfrak{S}_2}^V \end{array} &)(\rho) = \mathcal{L}\,({\mathfrak{S}_1}^V)(\rho) \cup \mathcal{L}\,({\mathfrak{S}_2}^V)(\rho)
\end{array}
$$

Fig. 10. Complexity Factors Extraction: $\mathcal{L}$

$init\,(g_i\rho)$, where g_i is a leaf goal and ρ is a valuation corresponding to one of the environments this leaf is evaluated for. The definition of $\mathcal{L}$ is shown in Fig. 10.

Now we can formulate the following main theorem that provides the principal recursive inequalities, extracted from the scheme for a given goal.

Theorem 4.1. *Let g be a goal, and let $\langle \epsilon,\, \varepsilon,\, n_{init}(g),\, V\rangle \;\vdash\; g \;\rightsquigarrow\; \mathfrak{S}^V$. Then*

$$
\begin{array}{l}
d\,(init\,(g\,\rho)) = \mathcal{D}\,(\mathfrak{S}^V)(\rho) + \Theta\,(1) \\
t\,(init\,(g\,\rho)) = \mathcal{T}\,(\mathfrak{S}^V)(\rho) + \Theta\,(\mathcal{D}\,(\mathfrak{S}^V)(\rho) \dot{-} \max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}(\mathfrak{S}^V)(\rho)} d\,(\langle g_i,\, e_i\rangle) + 1)
\end{array}
$$

being considered as functions on $\rho\colon V \to T_\varnothing$

The theorem allows us to extract two inequalities (upper and lower bounds) for both factors with a multiplicative constant that is the same for all valuations.

For our example, we can extract the following recursive inequalities from the scheme in Fig. 8. For presentation purposes, we will not show valuation in inequalities explicitly, but instead show the ground values of grounded variables

(using variables in bold font) that determine each valuation. We can do such a simplification for any concrete relation.

$$d\,(q^{app}\,(\mathbf{a},\mathbf{b})) = (1 + \sum_{\mathbf{a}=\mathtt{Nil}} 1) + (1 + \sum_{\mathbf{h},\mathbf{t}:\mathbf{a}=\mathtt{Cons(h,\ t)}} (d\,(q^{app}\,(\mathbf{t},\mathbf{b})) + \sum_{\mathbf{tb}:(\mathbf{t},\mathbf{b},\mathbf{tb})\in[\![\mathtt{append}^o]\!]} 1)) + \Theta\,(1)$$

$$\begin{aligned} t\,(q^{app}\,(\mathbf{a},\mathbf{b})) = (1 + \sum_{\mathbf{a}=\mathtt{Nil}} 1) + (1 + \sum_{\mathbf{h},\mathbf{t}:\mathbf{a}=\mathtt{Cons(h,\ t)}} (t\,(q^{app}\,(\mathbf{t},\mathbf{b})) + \sum_{\mathbf{tb}:(\mathbf{t},\mathbf{b},\mathbf{tb})\in[\![\mathtt{append}^o]\!]} 1)) \quad + \\ \Theta((1 + \sum_{\mathbf{a}=\mathtt{Nil}} 1) + (1 + \sum_{\mathbf{h},\mathbf{t}:\mathbf{a}=\mathtt{Cons(h,\ t)}} (d\,(q^{app}\,(\mathbf{t},\mathbf{b})) + \sum_{\mathbf{tb}:(\mathbf{t},\mathbf{b},\mathbf{tb})\in[\![\mathtt{append}^o]\!]} 1)) \quad - \\ \max_{\substack{\mathbf{h},\mathbf{t},\mathbf{tb}:\mathbf{a}=\mathtt{Cons(h,\ t)}\wedge \\ (\mathbf{t},\mathbf{b},\mathbf{tb})\in[\![\mathtt{append}^o]\!]}} \{d\,(init\,(ab \equiv \mathbf{b})), d\,(init\,(ab \equiv \mathtt{Cons(h,\ tb)}))\} + 1) \end{aligned}$$

Automatically extracted recursive inequalities, as a rule, are cumbersome, but they contain all the information on how scheduling affects the complexity. Often they can be drastically simplified by using metatheory-level reasoning.

For our example, we are only interested in the case when substituted values represent some lists. We thus perform the usual for lists case analysis considering the first list empty or non-empty. We can also notice that the excluded summand equals one. So we can rewrite the inequalities in the following way:

$$\begin{aligned} d\,(q^{app}\,(\mathtt{Nil},\mathbf{b})) &= \Theta\,(1) \\ d\,(q^{app}\,(\mathtt{Cons(h,\ t)},\mathbf{b})) &= d\,(q^{app}\,(\mathbf{t},\mathbf{b})) + \Theta\,(1) \\ t\,(q^{app}\,(\mathtt{Nil},\mathbf{b})) &= \Theta\,(1) \\ t\,(q^{app}\,(\mathtt{Cons(h,\ t)},\mathbf{b})) &= t\,(q^{app}\,(\mathbf{t},\mathbf{b})) + \Theta\,(d\,(q^{app}\,(\mathbf{t},\mathbf{b}))) \end{aligned}$$

These trivial linear inequalities can be easily solved:

$$\begin{aligned} d\,(q^{app}\,(\mathbf{a},\mathbf{b})) &= \Theta\,(len\,(\mathbf{a})) \\ t\,(q^{app}\,(\mathbf{a},\mathbf{b})) &= \Theta\,(len^2\,(\mathbf{a})) \end{aligned}$$

In this case, scheduling makes a big difference and changes the asymptotics. Note, we expressed the result using notions from metatheory (len for the length of the list represented by a term).

In contrast, if we consider the optimal definition $\mathtt{append}^o_{opt}$ the analysis of the call $q^{app\text{-}opt}\,(\mathbf{a},\mathbf{b}) = init\,(\mathtt{append}^o_{opt}\,\mathbf{a}\,\mathbf{b}\,ab)$ is analogous, but among the candidates for exclusion there is the value $d\,(q^{app\text{-}opt}\,(\mathbf{t},\mathbf{b}))$ since the recursive call is placed in a leaf. So the last simplified recursive approximation is the following (the rest is the same as in our main example):

$$t\,(q^{app\text{-}opt}\,(\mathtt{Cons\ (h,\ t)},\mathbf{b})) = t\,(q^{app\text{-}opt}\,(\mathbf{t},\mathbf{b})) + \Theta\,(1)$$

So in this case the complexity of both factors is linear on $len\,(\mathbf{a})$.

5 Evaluation

The theory we have built was so far applied to only one relation — $\mathtt{append}^o$ — which we used as a motivating example. With our framework, it turned out to be possible to explain the difference in performance between two nearly identical implementations, and the difference — linear vs. quadratic asymptotic complexity — was just as expected from the experimental performance evaluation. In this section, we present some other results of complexity estimations and discuss the adequacy of these estimations w.r.t. the real miniKanren implementations.

	d	t		d	t
$\texttt{append}^o$ **a b** ab	$len\,(\mathbf{a})$	$len^2\,(\mathbf{a})$	$\texttt{plus}^o$ **n m** r	$\lvert\mathbf{n}\rvert$	$\lvert\mathbf{n}\rvert$
$\texttt{append}^o_{opt}$ **a b** ab	$len\,(\mathbf{a})$	$len\,(\mathbf{a})$	$\texttt{plus}^o$ **n** m **r**	$\min\{\lvert\mathbf{n}\rvert, \lvert\mathbf{r}\rvert\}$	$\min\{\lvert\mathbf{n}\rvert, \lvert\mathbf{r}\rvert\}$
$\texttt{append}^o_{opt}$ a b **ab**	$len\,(\mathbf{ab})$	$len\,(\mathbf{ab})$	$\texttt{plus}^o$ n m **r**	$\lvert\mathbf{r}\rvert$	$\lvert\mathbf{r}\rvert$
$\texttt{revers}^o$ **a** r	$len^2\,(\mathbf{a})$	$len^3\,(\mathbf{a})$	$\texttt{mult}^o$ **n m** r	$\lvert\mathbf{n}\rvert \cdot \lvert\mathbf{m}\rvert$	$\lvert\mathbf{n}\rvert^2 \cdot \lvert\mathbf{m}\rvert$
$\texttt{revers}^o$ a **r**	$len^2\,(\mathbf{r})$	$len^2\,(\mathbf{r})$	$\texttt{mult}^o$ (S n) (S m) **r**	$\lvert\mathbf{r}\rvert^2$	$\lvert\mathbf{r}\rvert^2$

Fig. 11. Derived d- and t-factors for some goals; $len\,(\bullet)$ stands for the length of a ground list, $\lvert\bullet\rvert$ — for the value of a Peano number, represented as ground term.

Derived complexity estimations for a few other relations are shown in Fig. 11. Besides concatenation, we deal with naive list reversing and Peano numbers addition and multiplication. We show both $d-$ and $t-$ factors since the difference between the two indicates the cases when scheduling strikes in. We expect that for simple relations like those presented the procedure of deriving estimations should be easy; however, for more complex ones the dealing with extracted inequalities may involve a non-trivial metatheory reasoning.

The justification of the adequacy of our complexity estimations w.r.t. the existing miniKanren implementations faces the following problem: it is not an easy task to separate the contribution of scheduling from other components of the search procedure — unification and occurs check. However, it is common knowledge among Prolog users that in practice unification takes a constant time almost always; some theoretical basis for this is given in [1]. There are some specifics of unification implementation in miniKanren. First, for the simplicity of backtracking in a non-mutable fashion triangular substitution [3] is used instead of the idempotent one. It brings in an additional overhead which is analyzed in some detail in [4], but the experience shows that in the majority of practical cases this overhead is insignificant. Second, miniKanren by default performs the "occurs check", which contributes a significant overhead and often subsumes the complexity of all other search components. Meanwhile, it is known, that occurs checks are rarely violated [2]. Having said this, we expect that in the majority of the cases the performance of miniKanren programs with the occurs check disabled are described by scheduling complexity alone. In particular, this is true for all cases in Fig. 11. To confirm the adequacy of our model we evaluated the running time of these and some other goals (under the conditions we've mentioned) and found that it confirms the estimations derived using our framework. The details of implementation, evaluation, and results can be found in an accompanying repository. [6]

6 Related Work

To our knowledge, our work is the first attempt of comprehensive time complexity analysis for interleaving search in miniKanren. There is a number of separate observations on how certain patterns in relational programming affect performance and a number of "rules of thumb" based on these observations [4]. Some papers [9, 12, 16] tackle specific problems with relational programming

[6] https://www.dropbox.com/sh/ciceovnogkeeibz/AAAoclpTSDeY3OMagOBJHNiSa?dl=0

using quantitative time measuring for evaluation. These approaches to performance analysis, being sufficient for specific relational problems, do not provide the general understanding of interleaving search and its cost.

At the same time complexity analysis was studied extensively in the broader context of logic programming (primarily, for Prolog). As one important motivation for complexity analysis is granularity control in parallel execution, the main focus was set on the automated approaches.

Probably the best known among them is the framework [7] for cost analysis of logic programs (demonstrated on plain Prolog), implemented in the system CASLOG. It uses data dependency information to estimate the sizes of the arguments and the number of solutions for executed atoms. These estimations are formulated as recursive inequalities (more precisely, as difference equations for upper bounds), which are then automatically solved with known methods. The time and space complexity are expressed using these estimations, the variations of this approach can provide both upper [7] and lower [8] bounds.

An alternative approach is suggested in [11] for symbolic analysis of logic programs (demonstrated in Prolog with cuts). It constructs symbolic evaluation graphs capturing grounding propagation and reduction of recursive calls to previous ones, in the process of construction some heuristic approximations are used. These graphs may look similar to the symbolic schemes described in the Section 4 at first glance, but there is a principal difference: symbolic graphs capture the whole execution with all invoked calls (using inverse edges to represent cycles with recursive calls), while our schemes capture only the execution inside the body of a specific relation (representing the information about internal calls in terms of denotational semantics). The graphs are then transformed into term rewriting systems, for which the problem of the complexity analysis is well-studied (specifically, AProVE tool is used).

While these two approaches can be seen as partial bases for our technique, they are mainly focused on how the information about the arguments and results of the evaluated clauses can be derived automatically, since the calculation of time complexity of SLD-resolution is trivial when this information is available. In contrast, we are interested in the penalty of non-trivial scheduling of relational calls under interleaving search, so we delegate handling the information about calls to the reasoning in terms of a specific metatheory.

7 Discussion and Future Work

The formal framework presented in this paper analyzes the basic aspects of scheduling cost for interleaving search strategy from the theoretical viewpoint. As we have shown, it is sufficiently powerful to explain some surprising asymptotic behaviour for simple standard programs in miniKanren, but the applicability of this framework in practice for real implementations of miniKanren requires further investigation. Two key aspects that determine practical applicability are the admissibility of the imposed requirements and the correspondence of specific miniKanren implementations to the reference operational semantics, which should be studied individually for each application. We see our work as

the ground for the future development of methods for analyzing the cost of interleaving search.

Our approach imposes three requirements on the analyzed programs: disjunctive normal form, uniqueness of answers, and grounding of relational calls. The first two are rather non-restrictive: DNF is equivalent to the description of relation as a set of Horn clauses in PROLOG, and the majority of well-known examples in miniKANREN are written in this or very similar form. Repetition of answers is usually an indication of a mistake in a program [4]. The groundness condition is more serious: it prohibits program execution from presenting infinitely many individual ground solutions in one answer using free variables, which is a useful pattern. At the same time, this requirement is not unique for our work (the framework for CASLOG system mentioned above imposes exactly the same condition) and the experience shows that many important kinds of programs satisfy it (although it is hard to characterize the class of such programs precisely). Relaxing any of these restrictions will likely mess up the current relatively compact description of symbolic execution (for the conditions on relational calls) or the form of the extracted inequalities (for the DNF condition).

Also, for now, we confine ourselves to the problem of estimating the time of the full search for a given goal. Estimating the time before the first (or some specific) answer is believed to be an important and probably more practical task. Unfortunately, the technique we describe can not be easily adjusted for this case. The reason for this is that the reasoning about time (scheduling time in particular) in our terms becomes non-compositional for the case of interrupted search: if an answer is found in some branch, the search is cut short in other branches, too. Dealing with such a non-compositionality is a subject of future research.

References

1. Albert, L., Casas, R., Fages, F.: Average-case analysis of unification algorithms. Theor. Comput. Sci. **113**(1), 3–34 (1993). https://doi.org/10.1016/0304-3975(93)90208-B, https://doi.org/10.1016/0304-3975(93)90208-B
2. Apt, K.R., Pellegrini, A.: Why the occur-check is not a problem. In: Bruynooghe, M., Wirsing, M. (eds.) Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP'92, Leuven, Belgium, August 26-28, 1992, Proceedings. Lecture Notes in Computer Science, vol. 631, pp. 69–86. Springer (1992). https://doi.org/10.1007/3-540-55844-6_128, https://doi.org/10.1007/3-540-55844-6_128
3. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 445–532. Elsevier and MIT Press (2001). https://doi.org/10.1016/b978-044450813-3/50010-2, https://doi.org/10.1016/b978-044450813-3/50010-2
4. Byrd, W.E.: Relational Programming in Minikanren: Techniques, Applications, and Implementations. Ph.D. thesis, USA (2009)
5. Byrd, W.E., Ballantyne, A., Rosenblatt, G., Might, M.: A unified approach to solving seven programming problems (functional pearl). Proc. ACM Program. Lang. **1**(ICFP), 8:1–8:26 (2017). https://doi.org/10.1145/3110252, https://doi.org/10.1145/3110252

6. Byrd, W.E., Holk, E., Friedman, D.P.: miniKanren, live and untagged: quine generation via relational interpreters (programming pearl). In: Danvy, O. (ed.) Proceedings of the 2012 Annual Workshop on Scheme and Functional Programming, Scheme 2012, Copenhagen, Denmark, September 9-15, 2012. pp. 8–29. ACM (2012). https://doi.org/10.1145/2661103.2661105, https://doi.org/10.1145/2661103.2661105
7. Debray, S.K., Lin, N.: Cost analysis of logic programs. ACM Trans. Program. Lang. Syst. **15**(5), 826–875 (1993). https://doi.org/10.1145/161468.161472, https://doi.org/10.1145/161468.161472
8. Debray, S.K., López-García, P., Hermenegildo, M.V., Lin, N.: Lower bound cost estimation for logic programs. In: Maluszynski, J. (ed.) Logic Programming, Proceedings of the 1997 International Symposium, Port Jefferson, Long Island, NY, USA, October 13-16, 1997. pp. 291–305. MIT Press (1997)
9. Donahue, E.: Guarded Fresh Goals: Dependency-Directed Introduction of Fresh Logic Variables. In: third miniKanren and Relational Programming Workshop (2021)
10. Friedman, D.P., Byrd, W.E., Kiselyov, O.: The reasoned schemer. MIT Press (2005)
11. Giesl, J., Ströder, T., Schneider-Kamp, P., Emmes, F., Fuhs, C.: Symbolic evaluation graphs and term rewriting: a general methodology for analyzing logic programs. In: Schreye, D.D., Janssens, G., King, A. (eds.) Principles and Practice of Declarative Programming, PPDP'12, Leuven, Belgium - September 19 - 21, 2012. pp. 1–12. ACM (2012). https://doi.org/10.1145/2370776.2370778, https://doi.org/10.1145/2370776.2370778
12. Jin, E., Rosenblatt, G., Might, M., Zhang, L.: Universal Quantification and Implication in miniKanren. In: third miniKanren and Relational Programming Workshop (2021)
13. Kiselyov, O., Shan, C., Friedman, D.P., Sabry, A.: Backtracking, interleaving, and terminating monad transformers: (functional pearl). In: Danvy, O., Pierce, B.C. (eds.) Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, ICFP 2005, Tallinn, Estonia, September 26-28, 2005. pp. 192–203. ACM (2005). https://doi.org/10.1145/1086365.1086390, https://doi.org/10.1145/1086365.1086390
14. Kosarev, D., Lozov, P., Boulytchev, D.: Relational synthesis for pattern matching. In: d. S. Oliveira, B.C. (ed.) Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, Fukuoka, Japan, November 30 - December 2, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12470, pp. 293–310. Springer (2020). https://doi.org/10.1007/978-3-030-64437-6_15, https://doi.org/10.1007/978-3-030-64437-6_15
15. Rozplokhas, D., Vyatkin, A., Boulytchev, D.: Certified semantics for relational programming. In: d. S. Oliveira, B.C. (ed.) Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, Fukuoka, Japan, November 30 - December 2, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12470, pp. 167–185. Springer (2020). https://doi.org/10.1007/978-3-030-64437-6_9, https://doi.org/10.1007/978-3-030-64437-6_9
16. Sandre, L., Zaidi, M., Zhang, L.: Relational Floating-Point Arithmetic. In: third miniKanren and Relational Programming Workshop (2021)

A Proofs

Lemma A.1. *Let C_f and C_g be real positive constants. If*

$$h(x) = f(x) + C_f + \Theta(g(x) + C_g)$$

and value $g(x)$ is positive for arbitrary x then

$$h(x) = f(x) + \Theta(g(x))$$

Proof. 1. For some real postive C_1: $h(x) \geq f(x) + C_f + C_1 \cdot (g(x) + C_g)$, thus $h(x) \geq f(x) + C_1 \cdot g(x)$
2. For some real postive C_2: $h(x) \leq f(x) + C_f + C_2 \cdot (g(x) + C_g)$, thus $h(x) \geq f(x) + (C_f + C_2 + C_2 \cdot C_g) \cdot g(x)$ □

Lemma A.2. *If*

$$h_1(x) = f_1(x) + \Theta(g_1(x))$$

and

$$h_2(x) = f_2(x) + \Theta(g_2(x))$$

then

$$(h_1(x) + h_2(x)) = (f_1(x) + f_2(x)) + \Theta(g_1(x) + g_2(x))$$

Proof. Both lower- and upper-bound constants are the sums of the two corresponding constants for the given approximations. □

Lemma A.3. *Let $g_1(x) \geq g_2(x) > 0$ for all x. If*

$$h(x) = f(x) + \Theta(g_1(x) + g_2(x))$$

then

$$h(x) = f(x) + \Theta(g_1(x))$$

Proof. 1. For some real postive C_1:

$$h(x) \geq f(x) + C_1 \cdot (g_1(x) + g_2(x)) \geq f(x) + C_1 \cdot g_1(x)$$

2. For some real postive C_2:
$$h(x) \leq f(x) + C_2 \cdot (g_1(x) + g_2(x)) \leq f(x) + 2C_2 \cdot g_1(x)$$

□

Lemma A.4. *If*

$$\langle g, e \rangle \xrightarrow{l} s'$$

then

$$\begin{aligned} d(\langle g, e \rangle) &= d(s') + 1 \\ t(\langle g, e \rangle) &= t(s') + 1 \end{aligned}$$

Proof. Immediately from the definitions of the estimated values. □

Lemma 3.1. *For any two states s_1 and s_2*

$$\begin{aligned} d(s_1 \oplus s_2) &= d(s_1) + d(s_2) \\ t(s_1 \oplus s_2) &= t(s_1) + t(s_2) + cost_{\oplus}(s_1 \oplus s_2) \end{aligned}$$

where $cost_{\oplus}(s_1 \oplus s_2) = \min\{2 \cdot d(s_1) - 1, 2 \cdot d(s_2)\}$

Proof. Induction on the sum of values $d(s_1) + d(s_2)$. Both equations easily hold for both the rules [DisjStop] and [DisjStep] if we apply the inductive hypothesis for the next state. □

Lemma 3.2. *For any state s and any goal g*

$$d(s \otimes g) = d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i \rangle) \qquad (\star)$$

$$t(s \otimes g) = t(s) + cost_{\otimes}(s \otimes g) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} (t(\langle g, a_i \rangle) + cost_{\oplus}(\langle g, a_i \rangle \oplus (s'_i \otimes g))) \qquad (\dagger)$$

where

$$\begin{aligned} cost_{\otimes}(s \otimes g) &= d(s) \\ s'_i &= \textit{the first state in the trace for } s \textit{ after} \\ &\quad \textit{a transition delivering the answer } a_i \end{aligned}$$

Proof. Induction on the value $d(s \times g)$. Both equations easily hold for all the rules [ConjStop], [ConjStopAns], [ConjStep], [ConjStepAns] if we apply the inductive hypothesis for the next state. □

Lemma 3.3.

$$t\left(s \otimes g\right) = t\left(s\right) + \left(\sum_{a_i \in \mathcal{T}r^{ans}(s)} t\left(\langle g,\, a_i \rangle\right) \right) + \Theta\left(d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) - \dot{\max}_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right)\right)$$

Proof. After unfolding the defintions and throwing out the identical parts we have the following statement:

$$\begin{aligned} & d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\left\{2 \cdot d\left(\langle g,\, a_i \rangle\right) - 1, 2 \cdot d\left(s'_i \otimes g\right)\right\} = \\ & = \Theta\left(d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) - \dot{\max}_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right)\right) \end{aligned}$$

If $\mathcal{T}r^{ans}(s)$ is empty, the statement is trivial.

If $\mathcal{T}r^{ans}(s)$ is not empty, $\dot{\max}$ turns into a simple max. We establish lower and upper bounds separately.

1. Lower bound. Let's show that

$$\begin{aligned} & d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\left\{2 \cdot d\left(\langle g,\, a_i \rangle\right) - 1, 2 \cdot d\left(s'_i \otimes g\right)\right\} \geq \\ & \geq d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) - \max_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) \end{aligned}$$

   First, we can decrease both arguments of min:

$$\begin{aligned} & d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\left\{2 \cdot d\left(\langle g,\, a_i \rangle\right) - 1, 2 \cdot d\left(s'_i \otimes g\right)\right\} \geq \\ & \geq d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\left\{d\left(\langle g,\, a_i \rangle\right), d\left(s'_i \otimes g\right)\right\} \end{aligned}$$

   Let's consider two cases.

   1.1 The minimum in this expression is always reached on the first argument. Then

$$\begin{aligned} & d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\left\{d\left(\langle g,\, a_i \rangle\right), d\left(s'_i \otimes g\right)\right\} = \\ & = d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) \geq \\ & \geq d\left(s\right) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) - \max_{a_i \in \mathcal{T}r^{ans}(s)} d\left(\langle g,\, a_i \rangle\right) \end{aligned}$$

   1.2 a_m is the first answer, such that the minimum for a_m is reached on the second argument. Then the answers up to a_m are sufficient to prove the bound.

$$
\begin{aligned}
&d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\{d(\langle g, a_i\rangle), d(s'_i \otimes g)\} \geq \\
&\geq d(s) + \sum_{a_i \in \{a_1,\ldots,a_m\}} \min\{d(\langle g, a_i\rangle), d(s'_i \otimes g)\} = \\
&= d(s) + \sum_{a_i \in \{a_1,\ldots,a_{m-1}\}} d(\langle g, a_i\rangle) + d(s'_m \otimes g) = \\
&= d(s) + \sum_{a_i \in \{a_1,\ldots,a_{m-1}\}} d(\langle g, a_i\rangle) + d(s'_m) + \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_1,\ldots,a_m\})} d(\langle g, a_i\rangle) \geq \\
&\geq d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i\rangle) - d(\langle g, a_m\rangle) \geq \\
&\geq d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i\rangle) - \max_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i\rangle)
\end{aligned}
$$

2. Upper bound. Let's show that

$$
\begin{aligned}
&d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\{2 \cdot d(\langle g, a_i\rangle) - 1, 2 \cdot d(s'_i \otimes g)\} \leq \\
&\leq 4 \cdot (d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i\rangle) - \max_{a_i \in \mathcal{T}r^{ans}(s)} d(\langle g, a_i\rangle))
\end{aligned}
$$

For the upper bound we replace every min with any of its arguments (the result can only increase). Let $a_m = \underset{a_i \in \mathcal{T}r^{ans}(s)}{argmax}\ d(\langle g, a_i\rangle)$. Then for the upper bound let's go with the second argument of min for a_m and with the first argument for all others.

$$
\begin{aligned}
&d(s) + \sum_{a_i \in \mathcal{T}r^{ans}(s)} \min\{2 \cdot d(\langle g, a_i\rangle) - 1, 2 \cdot d(s'_i \otimes g)\} \leq \\
&\leq d(s) + 2 \cdot d(s'_m \otimes g) + \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_m\})} (2 \cdot d(\langle g, a_i\rangle)) \leq \\
&\leq d(s) + 2 \cdot d(s'_m) + 2 \cdot \sum_{a_i \in (\substack{\mathcal{T}r^{ans}(s) \setminus \\ \{a_1,\ldots,a_m\}})} d(\langle g, a_i\rangle) + 2 \cdot \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_m\})} d(\langle g, a_i\rangle) \leq \\
&\leq d(s) + 2 \cdot d(s) + 2 \cdot \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_m\})} d(\langle g, a_i\rangle) + 2 \cdot \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_m\})} d(\langle g, a_i\rangle) \leq \\
&\leq 4 \cdot d(s) + 4 \cdot \sum_{a_i \in (\mathcal{T}r^{ans}(s) \setminus \{a_m\})} d(\langle g, a_i\rangle)
\end{aligned}
$$

□

Lemma A.8. *Let $s = ((s_0 \otimes g_1) \cdots \otimes g_k)$ and let A_i be a set of all answers that are passed to g_i, i.e.*

$$A_1 = \mathcal{T}r^{ans}(s_0)$$
$$A_{i+1} = \bigcup_{a \in A_i} \mathcal{T}r^{ans}(\langle g_i,\, a\rangle)$$

Then

$$d\,(s) = \quad d\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} d\,(\langle g_i,\, a\rangle)$$

$$t\,(s) = \quad t\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} t\,(\langle g_i,\, a\rangle) +$$
$$+ \Theta\,(\; d\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} d\,(\langle g_i,\, a\rangle) - \dot{\max_{a \in A_k}}\, d\,(\langle g_k,\, a\rangle))$$

Proof. First we show that for all i, $A_{i+1} = \mathcal{T}r^{ans}(((s_0 \otimes g_1) \cdots \otimes g_i))$ (it's a simple induction on i and then on the trace).

Then we can prove the statement by induction on k. We unfold $d\,(s \otimes g_{k+1})$ and $t\,(s \otimes g_{k+1})$ using equations in Lem. 3.2. Then we rewrite $d\,(s)$ and $t\,(s)$ in these equations with inductive hypothesis. For $d\,(s \otimes g_{k+1})$ we get exactly the equation we need. For $t\,(s \otimes g_{k+1})$ we have a sum of the following parts (first two from the unfolding of $t\,(s)$, last two from the rest of the equation for $t\,(s \otimes g_{k+1})$):

1. $t\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} t\,(\langle g_i,\, a\rangle)$
2. $\Theta\,(d\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} d\,(\langle g_i,\, a\rangle) - \dot{\max_{a \in A_k}}\, d\,(\langle g_k,\, a\rangle))$
3. $\sum_{a \in A_{k+1}} t\,(\langle g_{k+1},\, a\rangle)$
4. $\Theta\,(d\,(s_0) + \sum_{1 \le i \le k} \sum_{a \in A_i} d\,(\langle g_i,\, a\rangle) + \sum_{a \in A_k} d\,(\langle g_{k+1},\, a\rangle) - \dot{\max_{a \in A_{k+1}}}\, d\,(\langle g_k,\, a\rangle))$

We can see that the second part is subsumed by the last part (by Lem. A.3). The rest gives exactly the equation we need. □

Here is the general definition of well-formedness of states from [15].

Definition A.1. *Well-formedness condition for extended states:*

- $\diamond$ *is well-formed;*
- $\langle g, \sigma, n\rangle$ *is well-formed iff* $\mathcal{FV}\,(g) \cup \mathcal{D}om\,(\sigma) \cup \mathcal{VR}an\,(\sigma) \subseteq \{\alpha_1, \ldots, \alpha_n\}$*;*
- $s_1 \oplus s_2$ *is well-formed iff* s_1 *and* s_2 *are well-formed;*
- $s \otimes g$ *is well-formed iff* s *is well-formed and for all leaf triplets* $\langle \_, \_, n\rangle$ *in* s *it is true that* $\mathcal{FV}\,(g) \subseteq \{\alpha_1, \ldots, \alpha_n\}$*.*

We will need Lem. 4.1 in the following generalized form.

Lemma A.9. *Let* $\pi\colon \{\alpha_1, \ldots, \alpha_N\} \to \{\alpha_1, \ldots, \alpha_{N'}\}$ *be an injective function on variables and let* R_π *be the following inductively defined relation on states:*

$$
\begin{array}{ll}
\Diamond R_\pi \Diamond & \\
\langle g, (\sigma, n)\rangle R_\pi \langle g', (\sigma', n')\rangle & \textit{iff } g\sigma\pi = g'\sigma' \\
(s_1 \oplus s_2) R_\pi (s_1' \oplus s_2') & \textit{iff } s_1 R_\pi s_1' \wedge s_2 R_\pi s_2' \\
(s \otimes g) R_\pi (s' \otimes g') & \textit{iff } s R_\pi s' \wedge \\
\quad \textit{for all substates } \langle g_i, (\sigma_i, n_i)\rangle \textit{ in } s & \\
\quad \textit{and corresponding substates } \langle g_i', (\sigma_i', n_i')\rangle \textit{ in } s', & \\
\quad g\sigma_i\pi = g'\sigma_i' &
\end{array}
$$

Then for any two well-formed states s *and* s'*, such that all counters occuring in* s *are less or equal than some* n *and all counters occuring in* s' *are less or equal than some* n' *and* $s R_\pi s'$ *for some injective function* $\pi\colon \{\alpha_1, \ldots, \alpha_n\} \to \{\alpha_1, \ldots, \alpha_n'\}$*,*

$$d(s) = d(s')$$

and

$$t(s) = t(s')$$

and there is a bijection b *between sets of answers* $\mathcal{T}r^{ans}(s)$ *and* $\mathcal{T}r^{ans}(s')$ *such that for any answer* $a = (\sigma_r, n_r) \in \mathcal{T}r^{ans}(s)$ *there is a corresponding answer* $b(a) = (\sigma_r', n_r') \in \mathcal{T}r^{ans}(s')$*, s.t.* $\sigma_r = \sigma\delta$ *for some* σ *that is a subtitution in some leaf substate of* s *and* $\sigma_r' = \sigma'\delta'$ *for* σ' *that is the substitutution of the corresponding leaf substate of* s' *and there is an injective function* $\pi_r\colon \{\alpha_1, \ldots, \alpha_{n_r}\} \to \{\alpha_1, \ldots, \alpha_{n_r'}\}$ *such that* $\pi_r \succ \pi$ *and* $\pi\delta' = \delta\pi_r$*.*

Proof. We prove it by induction on the length of the trace for s; simultaneously we prove that the next states in the traces for s and s' also satisfy the relation R_{π_r} for some π_r (s.t. $\pi_r \succ \pi$). The equalities $d(s) = d(s')$ and $t(s) = t(s')$ are obvious in this induction, because states in the relation R_π always have the same form (and therefore the same left height). To prove the fact about the bijection between answers and the fact that the relation holds for the next states we conduct an internal induction on the relation of operational semantics step. When we move through the introduction of the fresh variable we extend the injective function changing the variable by a binding between new fresh variables. In the base case of unification, when we extend the substitutions by the most general unifiers, we have the fact about the bijection between the sets of answers (singleton in this case) for the same injective renaming function by definition of the unification algorithm (we may change the names of variables before the unification and the result will be the same as if we do it after the unification):

$$\pi\, mgu\,(t_1\sigma\pi, t_2\sigma\pi) = mgu\,(t_1\sigma, t_2\sigma)\,\pi$$

For the case when we incorporate the answer obtained at this step in the next state (in rules [ConjStopAns] and [ConjStepAns]) we use the statement about the bijection between the sets of answers from the inductive hypothesis to prove that the next states satisfy the relation:

$$g\sigma\delta\pi_r = g\sigma\pi\delta' = g'\sigma'\delta'$$

All other cases naturally follow from inductive hypotheses. □

Lemma 4.1. *Let $s = \langle g, (\sigma, n)\rangle$ and $s' = \langle g', (\sigma', n')\rangle$ be two well-formed states. If there exists a bijective substitution $\pi\colon FV\,(g\sigma) \to FV\,(g'\sigma')$ such that $g\sigma\pi = g'\sigma'$, then $d\,(s) = d\,(s')$ and $t\,(s) = t\,(s')$.*

Proof. It is a special case of Lem. A.9. □

Lemma A.11. *Let $\langle g_0, (\sigma_0, n_0)\rangle$ be a leaf state. Then for every answer (σ', n') in $\mathcal{T}r^{ans}(\langle g_0, (\sigma_0, n_0)\rangle)$, $\sigma' = \sigma_0\delta$ for some substitution δ, such that $\mathcal{D}om\,(\delta) \cup \mathcal{VR}an\,(\delta) \subset FV\,(g_0\sigma_0) \cup \{\alpha_i \mid i > n_0\}$.*

Proof. First, we need some notions to generalize the statement.

Let $ENV\,(s)$ be the set of environments that occur in the given state.

$$\begin{array}{ll} ENV\,(\Diamond) & = \varnothing \\ ENV\,(\langle g, e\rangle) & = \{e\} \\ ENV\,(s_1 \oplus s_2) & = ENV\,(s_1) \cup ENV\,(s_2) \\ ENV\,(s \otimes g) & = ENV\,(s) \end{array}$$

Now, let's generalize the set of variables updated by answers from the statement to an arbitrary state.

$$\begin{array}{ll} \Delta\,(\Diamond) & = \varnothing \\ \Delta\,(\langle g, (\sigma, n)\rangle) & = FV\,(g\sigma) \cup \{\alpha_i \mid i > n\} \\ \Delta\,(s_1 \oplus s_2) & = \Delta\,(s_1) \cup \Delta\,(s_2) \\ \Delta\,(s \otimes g) & = \Delta\,(s_1) \cup \bigcup\limits_{(\sigma, n) \in ENV\,(s)} FV\,(g\sigma) \end{array}$$

Now we can generalize the statement but for one semantical step only: if $s \xrightarrow{l} s'$, then the following three conditions hold:

1. $\Delta\,(s) \supset \Delta\,(s')$

2. If $l = (\sigma', n')$ then there exists $(\sigma, n) \in ENV(s)$ and substitution δ, such that $\sigma' = \sigma\delta$ and $n' = n$ and $\mathcal{D}om(\delta) \cup \mathcal{VR}an(\delta) \subset \Delta(s)$
3. For any $(\sigma', n') \in ENV(s')$ there exists $(\sigma, n) \in ENV(s)$ and substitution δ, such that $\sigma' = \sigma\delta$ and $n' \geq n$ and $\mathcal{D}om(\delta) \cup \mathcal{VR}an(\delta) \subset \Delta(s)$

We prove it by the induction on semantical step relation (we have to prove all three conditions simultaneously).

1. The first condition is simple for the steps from leaf goals: the counters of occupied variables can only increase and the sets of free variables of subgoals can only decrease (except for the case of fresh variable introduction, where a new free variable appears, but it is greater than the counter); in case of invocation we use the fact that the body of any relation is closed (there are no free variables except for the arguments). For the [ConjStopAns] rule we use the second condition: the next step has the environment that updates one of the environments in s, so it does not introduce new variables in Δ and the counter also may only increase. For the [ConjStep] rule we use the third condition: all substitutions from environments of updated state after application to a goal do not introduce new variables in Δ, the rest is handled by the inductive hypothesis. For the [ConjStepAns] rule we combine two previous arguments and for other cases the first condition is obvious from the inductive hypothesis.
2. The second condition needs to be proven only for the [UnifySuccess] rule (where it follows from the properties of the unification algorithm) and for the rules [DisjStop] and [DisjStep] (where it is obvious from the inductive hypothesis).
3. The third condition follows simply in all cases from the inductive hypothesis and from the second condition (for the rules [ConjStep] and [ConjStepAns] where the answer is incorporated in the next state).

Now, the statement of the lemma follows from the generalized statement for one step: at each step substitutions in the answers and in the next step are composed with some additional substitutions that manipulate with only variables from the set Δ for this step, which is a subset of the set Δ in the beginning, which is exactly what we need. □

Lemma 4.2. *Let t_1, t_2 be terms, $V \subset \mathcal{A}$ and $\rho\colon V \to \mathcal{T}_\varnothing$ be a valuation. If $mgu(t_1, t_2) = \delta$ and $U = \boldsymbol{upd}(V, \delta)$ then $t_1\rho$ and $t_2\rho$ are unifiable iff there is some $\rho'\colon U \to \mathcal{T}_\varnothing$ such that $\rho' \succ \rho$ and $\forall(y = t) \in \boldsymbol{constr}(\delta, U) : \rho'(y) = t\rho'$. In such case ρ' is unique and $\rho \circ mgu(t_1\rho, t_2\rho) = \delta \circ \rho'$ up to alpha-equivalence (e.g. there exists a bijective substitution $\pi : FV(t_1) \to FV(t_2)$, s.t. $\rho \circ mgu(t_1\rho, t_2\rho) = \delta \circ \rho' \circ \pi$).*

Proof.

From the corectness of the Robinson's unification algoithm we know that a substitution unifies the pair of terms (t_1, t_2) iff it unifies all pairs of terms from the set $\{(x, \delta(x)) \mid x \in \mathcal{D}om(\delta)\}$ (because we obtain δ from the pair (t_1, t_2) by Robinson's algorithm that maintains equivalent unification problem).

First, let's notice that similarly a substitution unifies the pair of terms $(t_1\rho, t_2\rho)$ iff it unifies all pairs of terms $T = \{(\rho(x), \delta(x)\rho) \mid x \in \mathcal{D}om(\delta)\}$: ν is such substitution iff $\rho\nu$ unifies the terms (t_1, t_2). And then also any most general unifier for $(t_1\rho, t_2\rho)$ is the most general unifier for T and vice versa (by definition).

So now we need to show that T is unifiable iff the unique ρ' from the statement of the lemma exists (and that the most general unifier for T can be defined with ρ' and δ). In both directions we will use the induction on the construction of the set of variables U, so lets consider the following sequence U_i: $U_0 = V$ and $U_{i+1} = \{U_i \cup \bigcup\limits_{x \in U_i} FV(\delta(x))\}$ (so U is U_l such that $U_l = U_{l+1}$).

1. Suppose there is a substitution τ that unifies all the terms in T. Let's show that there is a unique ρ' such that $\rho' \succ \rho$ and $\forall(y, t) \in \mathbf{constr}(\delta, U)$, $\rho'(y) = t\rho'$.
   We know that $\rho(x)\tau = \delta(x)\rho\tau$ for all $x \in \mathcal{D}om(\delta)$. We need the same condition for ρ' for all $x \in U \cap \mathcal{D}om(\delta)$. We can now show by induction on i that for all variables $x \in U_i$ $(i \geq 1)$ the value $\tau(x)$ is ground and uniquely defined for a given ρ, so they can be taken as values of ρ' on variables from $U \setminus V$ and they are the only possible values. First, look at a pair $(\rho(x), \delta(x)\rho)$ in T for some $x \in V$. We know that $\rho(x)\tau = \delta(x)\rho\tau$ and the term on the lhs is ground and uniquely defined by ρ. So the values of τ for all free variables of $\delta(x)\rho$ are ground and uniquely defined by ρ, too. If we do it for all such pairs in T we will get the statement for U_1, then we can repeat this reasoning by induction for all U_i.
2. Now suppose there is ρ' such that $\rho' \succ \rho$ and $\forall(y, t) \in \mathbf{constr}(\delta, U), \rho'(y) = t\rho'$. Let's construct the most general unifier for T using the Robinson's algorithm.
   Let's split T on $T_U = \{(\rho(x), \delta(x)\rho) \mid x \in U \cap \mathcal{D}om(\delta)\}$ and $T_{-U} = \{(\rho(x), \delta(x)\rho) \mid x \in \mathcal{D}om(\delta) \setminus U\}$. We will be applying rules from the Robinson's algorithm to T for pairs of terms from T_U.
   First, let's look at some pair $(\rho(x), \delta(x)\rho)$ for $x \in V \cap \mathcal{D}om(\delta)$. By definition of ρ' we have $\rho(x)\rho' = \delta(x)\rho\rho'$. The first term in the pair is ground, the second one may contain free variables (then they are variables from U_1). If the second term is ground, too, they are equal and we can delete this pair. Otherwise, using decomposition rule we decompose this pair to pairs of terms with second term being variable. After this, we will have pairs $(\rho'(y), y)$ for all $y \in FV(\delta(x)\rho)$. After that we do it for all such pairs for all variables from $V \cap \mathcal{D}om(\delta)$, this pairs will turn into swapped bindings $(\rho'(y), y)$ for all $y \in U_1$ (maybe with repetitions). We then can discard the duplicates, swap the elements and apply this bindings in the rest of T. Now all the pairs $(\rho(x), \delta(x)\rho)$ for $x \in (U_1 \setminus V) \cap \mathcal{D}om(\delta)$ after substitution have ground terms

as the left term and we can repeat the transformation for all these pairs. We can repeat this process by induction on i for $x \in U_i$, until U_i becomes equal to U.
After this application of rules all pairs from T_U are decomposed and turned into the substitution (as a set of bindings) $\rho' \restriction_{U \setminus V}$. On the other hand the pairs from T_{-U} are not decomposed, just applied substitutions to them so every pair from this part still has some variable z as the first term (because $z \notin U$) and term $\delta(z)\rho(\rho' \restriction_{U \setminus V})$ as the second term. So we can see that we turned the set of terms T into the substitution $(\delta \restriction_{\mathcal{D}om(\delta) \setminus U} \rho') \restriction_{\mathcal{D}om(\delta) \setminus V}$ which equals $(\delta\rho') \restriction_{\mathcal{D}om(\delta) \setminus V}$ because ρ' unifies bindings in δ, so this substitution is the most general unifier for T and therefore for terms $t_1\rho$ and $t_2\rho$. Then this substitution is alpha-equivalent to $mgu(t_1\rho, t_2\rho)$ (because most general unifiers are unique up to alpha-equivalence). So if we take $\rho mgu(t_1\rho, t_2\rho)$, we will get substitution alpha-equivalent to the substitution $\rho((\delta\rho') \restriction_{\mathcal{D}om(\delta) \setminus V})$ which equals to the substitution $\delta\rho'$ (obviously separately for variables from V andfrom outside V). □

Theorem 4.1. *Let g be a goal, and let $\langle \epsilon, \varepsilon, n_{init}(g), V \rangle \vdash g \rightsquigarrow \mathfrak{S}^V$. Then*

$$d(init(g\,\rho)) = \mathcal{D}(\mathfrak{S}^V)(\rho) + \Theta(1)$$
$$t(init(g\,\rho)) = \mathcal{T}(\mathfrak{S}^V)(\rho) + \Theta(\mathcal{D}(\mathfrak{S}^V)(\rho) \dot{-} \max_{\langle g_i, e_i \rangle \in \mathcal{L}(\mathfrak{S}^V)(\rho)} d(\langle g_i, e_i \rangle) + 1)$$

being considered as functions on $\rho\colon V \to T_\varnothing$

Proof.

1. Suppose we have proven this statement for $g \in C_{nf}$. Let's show it holds for $g \in D_{nf}$.
   1.1 First, we prove it for $g \in F_{nf}$ by induction on the goal. After unfolding each fresh constructor we get a goal with the same scheme. Also going through each fresh constuct increases the values of $d(\cdot)$ and $t(\cdot)$ by 1 by Lem. A.4, this additional constant can be deleted by Lem. A.1.
   1.2 Now we prove it for $g \in D_{nf}$ by induction on the goal. Let $g = g_1 \vee g_2$. Let ${\mathfrak{S}_1}^V$ be ${\mathfrak{S}_2}^V$ be children of the root in $\mathfrak{S}^V$.
   By Lem. A.4 and Lem. 3.1 and Lem. 4.1 we have the following equations:

$$d(\langle g_1 \vee g_2, e_{init} \rangle) = d(\langle g_1, e_{init} \rangle) + d(\langle g_2, e_{init} \rangle) + 1$$

$$\begin{aligned} t(\langle g_1 \vee g_2, e_{init} \rangle) &= t(\langle g_1, e_{init} \rangle) + t(\langle g_2, e_{init} \rangle) \\ &\quad + \min(2d(\langle g_1, e_{init} \rangle) - 1, 2d(\langle g_2, e_{init} \rangle)) + 1 \end{aligned}$$

   After rewriting the right part with inductive hypotheses (combining them using Lem. A.1 and Lem. A.2) we get the following approximations.

$$d\left(\langle g_1\rho \vee g_2\rho,\, e_{init}\rangle\right) = \mathcal{D}({\mathfrak{S}_1}^V)(\rho) + \mathcal{D}({\mathfrak{S}_2}^V)(\rho) + \Theta\left(1\right)$$

$$\begin{aligned} t\left(\langle g_1\rho \vee g_2\rho,\, e_{init}\rangle\right) = {} & \mathcal{T}({\mathfrak{S}_1}^V)(\rho) + \mathcal{T}({\mathfrak{S}_2}^V)(\rho) + \\ & \Theta\left(\min\left(\mathcal{D}({\mathfrak{S}_1}^V)(\rho), \mathcal{D}({\mathfrak{S}_2}^V)(\rho)\right) + \right. \\ & + (\mathcal{D}({\mathfrak{S}_1}^V)(\rho) - \dot{\max_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)) + \\ & \left. + (\mathcal{D}({\mathfrak{S}_2}^V)(\rho) - \dot{\max_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)) + 1\right) \end{aligned}$$

For $d\left(\cdot\right)$ it is exactly what we need.

W.l.o.g. let's suppose the minimum in the approximation for $t\left(\cdot\right)$ is achieved at the first argument.

Let's consider two cases: which of $\dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_l}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)$ is the maximal leaf for the whole scheme.

1.2.1 Suppose $\dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right) \leq \dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)$.

Then we can absorb the summand $(\mathcal{D}({\mathfrak{S}_1}^V)(\rho) - \dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right))$ under Θ by the larger summand $\mathcal{D}({\mathfrak{S}_1}^V)(\rho)$ (which came from min) by Lem. A.3. We get the following approximation which is exactly what we need:

$$\begin{aligned} t\left(\langle g_1\rho \vee g_2\rho,\, e_{init}\rangle\right) = {} & \mathcal{T}({\mathfrak{S}_1}^V)(\rho) + \mathcal{T}({\mathfrak{S}_2}^V)(\rho) + \\ & \Theta\left(\mathcal{D}({\mathfrak{S}_1}^V)(\rho) + \right. \\ & \left. + (\mathcal{D}({\mathfrak{S}_2}^V)(\rho) - \dot{\max_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)) + 1\right) \end{aligned}$$

1.2.2 Suppose $\dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right) > \dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)$.

Then we establish the lower and the upper bounds separately.

1.2.2.1 For the lower bound we again first absorb the summand $(\mathcal{D}({\mathfrak{S}_1}^V)(\rho) - \dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right))$ by Lem. A.3 to get the following.

$$\begin{aligned} t\left(\langle g_1\rho \vee g_2\rho,\, e_{init}\rangle\right) \geq {} & \mathcal{T}({\mathfrak{S}_1}^V)(\rho) + \mathcal{T}({\mathfrak{S}_2}^V)(\rho) + \\ & C_1 \cdot (\mathcal{D}({\mathfrak{S}_1}^V)(\rho) + \\ & + (\mathcal{D}({\mathfrak{S}_2}^V)(\rho) - \dot{\max_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right)) + 1) \end{aligned}$$

Then replace $(-\dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right))$ by $(-\dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_1}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right))$ which is smaller by assumption.

1.2.2.2 For the upper bound we first replace $\mathcal{D}({\mathfrak{S}_1}^V)(\rho)$ that came form min by $\mathcal{D}({\mathfrak{S}_2}^V)(\rho)$ which is larger by the assumption and then absorb the summand $(\mathcal{D}({\mathfrak{S}_2}^V)(\rho) - \dot{\max\limits_{\langle g_i,\, e_i\rangle \in \mathcal{L}({\mathfrak{S}_2}^V)(\rho)}} d\left(\langle g_i,\, e_i\rangle\right))$ by it.

2. Now let's prove the statement of the theorem for $g \in C_{nf}$.
Let $g = ((g_1 \wedge g_2) \wedge \dots) \wedge g_k$ with $g_i \in B_{nf}$.

First, notice that the state $\langle g,\, e_{init}\rangle$ is transformed into the state $((\langle g_1,\, (\varepsilon,\, n_{init}(g))\rangle \otimes g_2) \otimes \dots) \otimes g_k$ after $(k-1)$ steps of turning conjunctions into $\otimes$-states. All states during this steps except the resulting one add $(k-1)$ to the value $d\,(\langle g,\, e_{init}\rangle)$ and $\dfrac{(k-1)(k-2)}{2}$ to the value $t\,(\langle g,\, e_{init}\rangle)$, we can hide this constants under Θ by Lem. A.3. Lem. A.8 gives us the approximations of the measures for the state $((\langle g_1,\, (\varepsilon,\, n_{init}(g))\rangle \otimes g_2) \otimes \dots) \otimes g_k$. To put it in a convenient form we will use the following definitions.

$$\begin{aligned} D\,(s, \epsilon) &= d\,(s) \\ D\,(s, g : \Gamma) &= d\,(s) + \sum_{a \in \mathcal{T}r^{ans}(s)} D\,(\langle g,\, a\rangle, \Gamma) \end{aligned}$$

$$\begin{aligned} T\,(s, \epsilon) &= t\,(s) \\ T\,(s, g : \Gamma) &= t\,(s) + \sum_{a \in \mathcal{T}r^{ans}(s)} T\,(\langle g,\, a\rangle, \Gamma) \end{aligned}$$

$$\begin{aligned} L\,(s, \epsilon) &= \{s\} \\ L\,(s, g : \Gamma) &= \bigcup_{a \in \mathcal{T}r^{ans}(s)} T\,(\langle g,\, a\rangle, \Gamma) \end{aligned}$$

Now, Lem. A.8 gives us the following approximations if we denote $g_2\rho : \dots g_k\rho$ by Γ.

$$d\,(\langle g\rho,\, e_{init}\rangle) = D\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma) + \Theta\,(1)$$

$$\begin{aligned} t\,(\langle g\rho,\, e_{init}\rangle) &= T\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma) + \\ &\quad \Theta\,(D\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma) - \max_{s \in L\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma)}\!{}^{\bullet}\, d\,(s) + 1) \end{aligned}$$

It's the approximation in the form required in the statement of the theorem. What remains to be proven are the following equalities:

$$\begin{aligned} D\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma) &= \mathcal{D}(\mathfrak{S}^V)(\rho) \\ T\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma) &= \mathcal{T}(\mathfrak{S}^V)(\rho) \\ \{d\,(s) \mid s \in L\,(\langle g_1\rho,\, (\varepsilon,\, n_{init}(g))\rangle, \Gamma)\} &= \{d\,(s) \mid s \in \mathcal{L}(\mathfrak{S}^V)(\rho)\} \end{aligned}$$

We can prove them by induction, but first we need to generalize the statement. Let g be a goal from B_{nf} and $\Gamma = g_1 : \dots : g_m : \epsilon$ — a sequence of goals from B_{nf}, σ be a substitution, n be a fresh variable counter such that the state $(((\langle g,\, (\sigma,\, n)\rangle \otimes g_1) \otimes \dots) \otimes g_m)$ is well-formed, and V be a subset of variables $\{\alpha_1, \dots, \alpha_n\}$. Then if

$$\langle \Gamma,\, \sigma,\, n,\, V\rangle \;\vdash\; g \;\rightsquigarrow\; \mathfrak{S}^V$$

then the following equalities hold for any $\rho\colon V \to \mathcal{T}_\varnothing$.

$$\begin{aligned} D\,(\langle g,\, (\sigma\rho,\, n)\rangle, \Gamma) &= \mathcal{D}(\mathfrak{S}^V)(\rho) \\ T\,(\langle g,\, (\sigma\rho,\, n)\rangle, \Gamma) &= \mathcal{T}(\mathfrak{S}^V)(\rho) \\ \{d\,(s) \mid s \in L\,(\langle g,\, (\sigma\rho,\, n)\rangle, \Gamma)\} &= \{d\,(s) \mid s \in \mathcal{L}(\mathfrak{S}^V)(\rho)\} \end{aligned}$$

(to get from this generalization to the equations above we should take $\sigma = \varepsilon$ and apply Lem. 4.1 to move ρ from environment to the goal).
We prove the generalized statement by the induction on Γ and considering cases when g is an equality or a relational call. The reasoning is exactly the same for all three notions $D(\cdot)$, $T(\cdot)$ and $L(\cdot)$, so we demonstrate only the proof of the equality between $D(\cdot)$ and $\mathcal{D}(\cdot)(\cdot)$.

2.1 Let $\Gamma = \epsilon$ and $g = (t_1 \equiv t_2)$.
In this case we have
$$d(\langle t_1 \equiv t_2, (\sigma\rho, n)\rangle) = d(\langle t_1\sigma\rho \equiv t_2\sigma\rho, e_{init}\rangle) = 1$$

2.2 Let $\Gamma = \epsilon$ and $g = (R^k(t_1, \ldots, t_k))$.
In this case we have
$$d(\langle R^k(t_1, \ldots, t_k), (\sigma\rho, n)\rangle) = d(\langle R^k(t_1\sigma\rho, \ldots, t_k\sigma\rho), e_{init}\rangle)$$
Which is true by Lem. 4.1.

2.3 Let $\Gamma = g' : \Gamma'$ and $g = (t_1 \equiv t_2)$.

2.3.1 If the terms $t_1\sigma$ and $t_2\sigma$ are non-unifiable, we have
$$d(\langle t_1 \equiv t_2, (\sigma\rho, n)\rangle) + \sum_{mgu(t_1\sigma\rho, t_2\sigma\rho) = \delta'} D(\langle g', (\sigma\rho\delta', n)\rangle, \Gamma') = 1$$
And it is obviously true because the sum is empty since the more specific terms there are non-unifiable also.

2.3.2 If they are unifiable, we have
$$d(\langle t_1 \equiv t_2, (\sigma\rho, n)\rangle) + \sum_{mgu(t_1\sigma\rho, t_2\sigma\rho) = \delta'} D(\langle g', (\sigma\rho\delta', n)\rangle, \Gamma') =$$
$$= 1 + \sum_{\substack{\rho' : U \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ \forall (y,t) \in Cs, \rho'(y) = t\rho'}} \mathcal{D}(\mathfrak{S}^U)(\rho')$$
where
$$\begin{aligned} \delta &= mgu(t_1\sigma, t_2\sigma) \\ U &= \mathbf{upd}(V, \delta) \\ Cs &= \mathbf{constr}(\delta, U) \end{aligned}$$
The left summands are obviously equal. The rest is basically covered by Lem. 4.2. By this lemma there exists a most general unifier δ' iff the required ρ' exists. So both sums are non-empty under the same conditions and have at most one summand (since ρ' is unique), and if it is the case these summands are equal by Lem. 4.2, the inductive hypothesis and the fact that the value D is stable w.r.t. renaming of variables (it is a generalization of Lem. 4.1 that follows simply from Lem. A.9):
$$\begin{aligned} &D(\langle g', (\sigma\rho\delta', n)\rangle, \Gamma') \overset{\text{Lem. 4.2}}{=} D(\langle g', (\sigma\delta\rho'\tau, n)\rangle, \Gamma') = \\ &= D(\langle g', (\sigma\delta\rho', n)\rangle, \Gamma') \overset{\text{ind.hyp.}}{=} \mathcal{D}(\mathfrak{S}^U)(\rho') \end{aligned}$$

2.4 Let $\Gamma = g' : \Gamma'$ and $g = (R^k\,(t_1,\,\ldots,\,t_k))$.
In this case we have

$$d\,(\langle R^k\,(t_1,\,\ldots,\,t_k),\,(\sigma\rho,\,n)\rangle) + \sum_{a \in \mathcal{T}r^{ans}(\langle R^k\,(t_1,\ldots,t_k),\,(\sigma\rho,\,n)\rangle)} D\,(\langle g',\,a\rangle, \Gamma') =$$

$$= d\,(\langle R^k\,(t_1\sigma\rho,\,\ldots,\,t_k\sigma\rho),\,e_{init}\rangle) + \sum_{\substack{\rho' :\, U \to \mathcal{T}_\varnothing \\ \rho' \succ \rho \\ (t_1\sigma\rho',\ldots,t_k\sigma\rho') \in [\![R^k]\!]}} \mathcal{D}(\mathfrak{S}^U)(\rho')$$

where

$$U = V \cup \textstyle\bigcup_i FV\,(t_i\sigma)$$

The left summands are equal by Lem. 4.1.
Each $a \in \mathcal{T}r^{ans}(\langle R^k\,(t_1,\,\ldots,\,t_k),\,(\sigma\rho,\,n)\rangle)$ has the form $(\sigma\rho\delta, n')$ for some δ and some $n' > n$ by Lem. A.9. All free variables of terms t_i are mapped into ground terms in each delta, because the relational call is grounding.
There is a bijection between the set of answers under the sum on the lhs and a set of suitable ρ' at the rhs: all ground values for the free variables that are consistent with the denotational semantics are present in some answer (because of completeness of the operational semantics w.r.t. to the denotational one) and in exactly one (because the call is answer-unique).
Now, we need to show that if we take any answer $(\sigma\rho\delta, n')$ and ρ' that corresponds to it by the described bijection then the values for them under the sums are equal. First, we need to replace $D\,(\langle g',\,(\sigma\rho\delta, n')\rangle, \Gamma')$ by $D\,(\langle g',\,(\sigma\rho\delta\ \restriction_U, n)\rangle, \Gamma')$, we can do it because all variables from $\mathcal{D}om\,(\delta) \setminus U$ have indices greater or equal than n (by Lem. A.11) and therefore these variables are not free variables of goals from $(g' : \Gamma')$, so the values of d on this goals will be the same for the restricted substitution (it follows simply from Lem. A.9). After this replacement we can directly apply the inductive hypothesis. □