ICGT'12, LNCS, 7562, pp. 294-310. 2012

A Trully Concurrent Semantics for the $\mathbb{K}$ Framework Based on Graph Transformations

Traian Florin Șerbănuță and Grigore Roșu

University of Illinois at Urbana-Champaign
{tserban2,grosu}@illinois.edu

Abstract. This paper gives a truly concurrent semantics with sharing of resources for the $\mathbb{K}$ semantic framework, an executable (term-)rewriting-based formalism for defining programming languages and calculi. Akin to graph rewriting rules, the $\mathbb{K}$ (rewrite) rules explicitly state what can be concurrently shared with other rules. The desired true concurrency is obtained by translating the $\mathbb{K}$ rules into a novel instance of term-graph rewriting with explicit sharing, and then using classical concurrency from the double-pushout (DPO) approach to graph rewriting. The resulting parallel term-rewriting relation is proved sound, complete, and serializable with respect to the jungle rewriting flavor of term-graph rewriting, and, therefore, also to term rewriting.

1 Introduction

There are several reasons for defining a truly concurrent semantics for a given model of computation. One reason is that specification languages based on truly concurrent models are more informative; e.g., testing sequences defining partial orderings may carry the same information as an exponentially larger number of interleaving traces. Another reason is that in truly concurrent models the existing fine parallelism of the application is fully specified. It is left to the implementor to take advantage of it by allocating concurrent events to different processors, or to partition events into coarser classes performed by a few concurrent processes. Finally, truly concurrent semantics carries extra information, being usually straightforward to recover interleaving semantics from it [10].

The $\mathbb{K}$ semantic framework [12,13] is a programming language definitional framework based on rewriting which attempts to bring together the strengths of existing frameworks (e.g., the chemical abstract machine (CHAM) [1], evaluation contexts [15], or continuations [4]) while avoiding their weaknesses. The $\mathbb{K}$ framework relies on computations, configurations, and $\mathbb{K}$ rules in giving semantics to programming language constructs. So far, $\mathbb{K}$ has been successfully used for defining several real-life programming languages, including C [3], Scheme [7], and the x86-TSO memory model [13].

Currently, computations and configurations are described as algebraic terms over a first-order signature, and the semantics of $\mathbb{K}$ definitions is given through their translation in rewriting logic [8] theories. Structuring execution configurations as terms is quite convenient, as first order signatures are quite intuitive, and

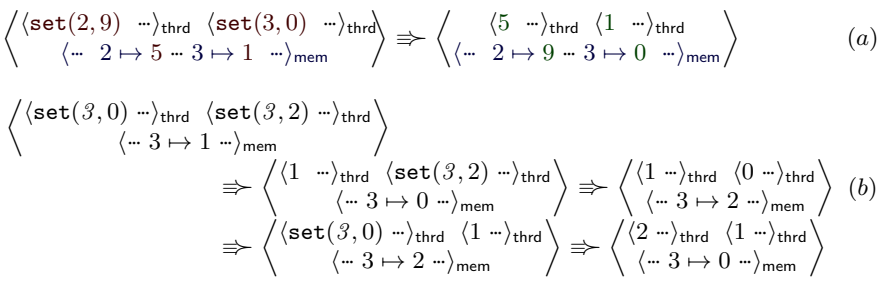


Fig. 1. Synchronous access of memory in a multithreaded environment: (a) concurrent writes, and (b) interleaving dataraces.

there is plenty of tool support for reasoning about first-order terms. Moreover, rewriting logic is generally appealing for defining truly concurrent systems, since rewrite rules can independently match and apply anywhere, unconstrained by the context [8]. However, although rewriting logic has proved successful in defining sequential programming languages as well as actor-like languages [9], it enforces that "the same object cannot be shared by two simultaneous rewrites" [9], i.e., rule instances are not allowed to overlap. Although there are good reasons for this choice, such as sufficing to capture concurrent synchronization like that of Petri Nets, this limitation enforces an interleaving semantics in situations where one may not want it, especially when describing systems which allow sharing of resources.

Consider a running configuration of a program where two threads are both ready to set the value of different memory locations, as in the left-hand side (lhs) of Figure 1(a). Assume the (single-threaded) semantics of the memory update construct `set` is to update the value in the memory and return the old value, like $\langle \texttt{set}(X, V') \cdots \rangle_{\mathsf{thrd}} \langle \cdots X \mapsto V \cdots \rangle_{\mathsf{mem}} \longrightarrow \langle V \cdots \rangle_{\mathsf{thrd}} \langle \cdots X \mapsto V' \cdots \rangle_{\mathsf{mem}}$. Then two instances of this rule (i.e., two threads attempting to concurrently update *distinct* memory locations) should be allowed to advance concurrently in one transition step as in Figure 1(a). In fact, two instances of such memory access rules should be forced to interleave *only if trying to concurrently access the same location, one of the accesses being a* `set`, as exemplified in Figure 1(b).

However, using the rewriting logic semantics associated to the rule above, the concurrent rewriting transition from Figure 1(a) would need to be interleaved, too. The reason is that the two instances of the rule overlap on the mem cell and on the algebraic constructs representing the cell composition operators. Nevertheless, it is worthwhile noting that the operators on which the rule instances overlap are only mentioned as context needed for the transition to apply, playing the same "gluing" role that interface graphs play in the DPO approach to graph transformations [2].

The special rewrite rules employed by the $\mathbb{K}$ framework, from here on named $\mathbb{K}$ rules, make this sharing of context explicit: rewrite rules are extended to allow specifying which parts of the matching pattern are effectively changed by a rule, allowing the rest to be shared with other rules. For example, the $\mathbb{K}$ rule

corresponding to the `set` rewrite rule presented above is:

$$\langle \underline{\texttt{set}(X, V')} \cdots \rangle_{\mathsf{thrd}} \ \langle \cdots X \mapsto \underline{V} \cdots \rangle_{\mathsf{mem}}$$
$$\qquad\qquad V \qquad\qquad\qquad\qquad V'$$

The intuition for the above rule is that, while the entire pattern of the top needs to be matched for the rule to apply, only the underlined `set` instruction and the value in the memory are actually changed by the rule, being replaced with the corresponding values below the line. This furthermore implies that the thrd and mem containers, along with the ellipses (specifying potential additional content), called the *read-only pattern* of the rule are only needed to specify the context in which the local transformations would apply, and thus can be shared with other concurrent instances of $\mathbb{K}$ rules.

Contributions This paper gives semantics to $\mathbb{K}$ rewriting through the help of graph rewriting, adapting existing representations of terms and rules as graph and graph rewrite rules to maximize their potential for concurrent application. The main result, Theorem 2, shows that $\mathbb{K}$ rewriting is *sound and complete* w.r.t. standard term rewriting, and that the concurrent application of $\mathbb{K}$ rules is *serializable*. Soundness means that applying one $\mathbb{K}$ rule can be simulated by applying its corresponding direct representation as a rewrite rule. Completeness means the converse, i.e., that one application of a term rewriting rule can be simulated by applying the corresponding $\mathbb{K}$ rule directly. Finally, the serialization result ensures that applying multiple $\mathbb{K}$ rules in parallel can be simulated by applying them one by one, obtaining an interleaving semantics for $\mathbb{K}$ rewriting through standard rewriting, which is one of the desirable goals for all truly concurrent models of computation [10]. Interestingly, a novel and unexpected acyclicity condition (presented in Section 5) was required to ensure serializability.

The remainder of the article is organized as follows. Section 2 formally defines $\mathbb{K}$ rules and relates them to term-rewrite rules. Section 3 recalls the basic definitions from the DPO approach to graph rewriting. Section 4 formalizes the encoding of terms with variables as graphs used by $\mathbb{K}$ graph rewriting. Section 5 presents the encoding of $\mathbb{K}$ rules as graph-rewriting rules, defines $\mathbb{K}$ graph rewriting as a term-graph rewriting formalism, and shows it admits parallel derivations which are serializable. Finally, Section 6 reflects $\mathbb{K}$ graph rewriting upon term rewriting proving soundness, completeness, and serialization of parallel derivations w.r.t. term-rewriting, and Section 7 concludes. An appendix with background material on jungle rewriting and proofs for all claimed results is included for reviewers' convenience; these are also available in the companion technical report [14].

2 $\mathbb{K}$ rules

The $\mathbb{K}$ rules describe how a term can be transformed into another term by altering some of its parts. They share the idea of match-and-replace of standard

term rewriting, but each $\mathbb{K}$ rule also specifies which part of the pattern is read-only. These read-only patterns are akin to the interfaces in graph rewriting [2], being used to glue together read-write patterns, that is, subparts to be rewritten. Moreover, through their variables, the read-only patters also provide information which can be used and shared by the read-write patterns.

A signature Σ is a pair (S, F) where S is a set of *sorts* and F is a set of operations $f : w \to s$, where f is an operation symbol, $w \in S^*$ is its arity, and $s \in S$ is its result sort. If w is the empty word ϵ then f is called a constant. T_Σ is the universe of (ground) terms over Σ and $T_\Sigma(\mathcal{X})$ is that of Σ-terms with variables from the S-sorted set $\mathcal{X}$. Given term $t \in T_\Sigma(\mathcal{X})$, let $vars(t)$ be the variables from $\mathcal{X}$ appearing in t. Given an ordered set of variables, $\mathcal{W} = \{\Box_1, \ldots, \Box_n\}$, named *context variables*, or *holes*, a $\mathcal{W}$-*context over* $\Sigma(\mathcal{X})$ (assume that $\mathcal{X} \cap \mathcal{W} = \emptyset$) is a term $C \in T_\Sigma(\mathcal{X} \cup \mathcal{W})$ in which each variable in $\mathcal{W}$ occurs once. The instantiation of a $\mathcal{W}$-context C with an n-tuple $\overline{t} = (t_1, \ldots, t_n)$, written $C[\overline{t}]$ or $C[t_1, \ldots, t_n]$, is the term $C[t_1/\Box_1, \ldots, t_n/\Box_n]$. One can regard $\overline{t}$ as a substitution $\overline{t} : \mathcal{W} \to T_\Sigma(\mathcal{X})$, defined by $\overline{t}(\Box_i) = t_i$, in which case $C[\overline{t}] = \overline{t}(C)$.

Definition 1. *A* $\mathbb{K}$ ***rule*** $\rho : (\forall \mathcal{X})\ k[\ L \Rightarrow \mathcal{R}\ ]$ *over a signature* $\Sigma = (S, F)$ *is a tuple* $(\mathcal{X}, k, L, \mathcal{R})$*, where:*

- $\mathcal{X}$ *is an* S*-sorted set, called the* ***variables*** *of the rule* ρ*;*
- k *is a* $\mathcal{W}$*-context over* $\Sigma(\mathcal{X})$*, called the* ***rule pattern****, where* $\mathcal{W}$ *are the* ***holes*** *of* k*;* k *can be thought of as the "read-only" part of* ρ*;*
- $L, \mathcal{R} : \mathcal{W} \to T_\Sigma(\mathcal{X})$ *associate to each hole in* $\mathcal{W}$ *its* ***original*** *and* ***replacement term****;* L*,* $\mathcal{R}$ *can be thought of as the "read/write" part of* ρ*.*

We may write $(\forall \mathcal{X})\ k[\ \underline{l_1}, \ldots, \underline{l_n}\ ]$ *instead of* $(\forall \mathcal{X})\ k[\ L \Rightarrow \mathcal{R}\ ]$ *whenever* $\mathcal{W} = \{\Box_1, \cdots, \Box_n\}$ *and* $L(\Box_i) = l_i$ *and* $\mathcal{R}(\Box_i) = r_i$*; this way, the holes are implicit and need not be mentioned.*

A set of $\mathbb{K}$ *rules is called a* $\mathbb{K}$ ***system****.*

The variables in $\mathcal{W}$ are only used to identify the positions in k where rewriting takes place; in practice we typically use the compact notation above, that is, underline the to-be-rewritten subterms in place and write their replacement underneath. When the set of variables $\mathcal{X}$ is clear, it can be omitted.

Given a $\mathbb{K}$ rule $\rho : (\forall \mathcal{X})\ k[\ L \Rightarrow \mathcal{R}\ ]$, its associated 0-*sharing* $\mathbb{K}$ *rule* is $\rho_0 : (\forall \mathcal{X})\ \Box[\ \underline{L(k)}\ ]$, which is a $\mathbb{K}$ rule specifying the same transformation but $\mathcal{R}(k)$ without sharing anything. It is relatively easy to see that one can associate to any $\mathbb{K}$ rule ρ as above a regular rewrite rule $K2R(\rho) = (\forall X)L(k) \to \mathcal{R}(k)$. This is to account for the fact that, when applied in a non-concurrent fashion, $\mathbb{K}$ rules must obey the standard rewriting semantics.

Conversely, given a rewrite rule $\rho : (\forall \mathcal{X})l \to r$, let $R2K(\rho)$ denote the 0-sharing $\mathbb{K}$ rule for which $K2R(R2K(\rho)) = \rho$, that is $(\forall \mathcal{X})\ \Box[\ \underline{l}\ ]$. For this reason, r we take the liberty to denote a 0-sharing $\mathbb{K}$ rule $\rho : (\forall \mathcal{X})\ \Box[\ \underline{l}\ ]$ by $l \to r$. r

Running Example Consider the following $\mathbb{K}$ system, where h is a ternary operation, g is binary, f is unary, 0, 1, a, b are constants, and x, y are variables:

$$(1)\ h(\frac{x}{g(x,x)}, y, \frac{1}{0}) \qquad (2)\ h(x, \frac{0}{1}, y) \qquad (3)\ a \to b \qquad (4)\ f(x) \to x$$

For each of the rules above, its corresponding parts according to Definition 1 are:

rule	X	$\mathcal{W}$	p	L	$\mathcal{R}$
(1)	$\{x, y\}$	$\{\Box_1, \Box_2\}$	$h(\Box_1, y, \Box_2)$	$\Box_1 \mapsto x\ ;\ \Box_2 \mapsto 1$	$\Box_1 \mapsto g(x,x)\ ;\ \Box_2 \mapsto 0$
(2)	$\{x\}$	$\{\Box\}$	$h(x, \Box, y)$	$\Box \mapsto 0$	$\Box \mapsto 1$
(3)	$\emptyset$	$\{\Box\}$	$\Box$	$\Box \mapsto a$	$\Box \mapsto b$
(4)	$\{x\}$	$\{\Box\}$	$\Box$	$\Box \mapsto f(x)$	$\Box \mapsto x$

Suppose now we want to rewrite term $h(f(a), 0, 1)$ using rules (1)–(4). Flattening these $\mathbb{K}$ rules into rewrite rules, in rewriting logic one can apply either rules (1), (3), and (4), or rules (2), (3), and (4) concurrently on the term $h(f(a), 0, 1)$, to obtain either $h(g(b, b), 0, 0)$ or $h(b, 1, 1)$, respectively. However, executing both rules (1) and (2) in parallel is impossible with the deduction rules of rewriting logic because "the same object cannot be shared by two simultaneous rewrites" [9], i.e., rule instances are not allowed to overlap. Nevertheless, as seen in the examples in Section 1, being able to apply rules like (1) and (2) concurrently is crucial for capturing the truly concurrent semantics of programming languages.

In spite of the fact that all four rule instances above overlap on the term $h(f(a), 0, 1)$, using $\mathbb{K}$ rules one can intuitively apply *all* four rules concurrently. First, note that rule (1) modifies the first and the third arguments of h regardless of the second argument, while rule (2) modifies the second argument of h regardless of its first and third arguments. Therefore, rules (1) and (2) share the top operator h and yield complementary changes on the original term, so they can safely apply their changes in parallel on term $h(f(a), 0, 1)$. Moreover, note that none of these rules relies on x being bound to $f(a)$, or what happens with $f(a)$ during their application. Therefore, we can rewrite the $f(a)$ that x points to in parallel with the application of rules (1) and (2). Using a similar argument, rules (3) and (4) can apply in parallel on $f(a)$ to rewrite it to b. Thus, all of the rules can apply in one parallel rewrite step on $h(f(a), 0, 1)$ and produce $h(g(b, b), 1, 0)$.

In the remainder of this paper we will formalize $\mathbb{K}$ term rewriting, i.e., rewriting using $\mathbb{K}$ rules, through an embedding into graph rewriting theory. The reasons for our choice are: (1) the intuition that the pattern k of a $\mathbb{K}$ rule is meant to be "shared" with competing concurrent rule instances is conceptually captured by the notion of interface graphs of graph rewrite rules in the double-pushout (DPO) algebraic approach to graph rewriting [2]; (2) (term) graph rewriting [5,11] was shown to be sound and complete for term rewriting, which we want to preserve for $\mathbb{K}$; and (3) the results in the DPO theory of graph rewriting show that if graph rule instances only overlap on the interface graphs, then they can be concurrently applied and the obtained rewrite step is serializable [6], which is also the desirable semantics for $\mathbb{K}$.

As our interests fall at the convergence of term-graph rewriting (for being sound and complete w.r.t. term rewriting) and the DPO approach to graph rewriting (for concurrency with sharing of context), the subsequent graph embedding of $\mathbb{K}$ rewriting can be seen as an extension (enhancing the concurrency, while conserving soundness and completeness) of jungle hypergraph rewriting [5].

The remainder of the paper describes $\mathbb{K}$ *graph rewriting*, which uses the same mechanisms and intuitions of jungle rewriting, but adapts the jungle term-graphs and graph-rewrite rules [5] to increase the potential for concurrency, both with sharing and without sharing of context.

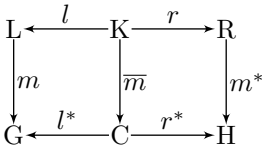
3 Graph Transformations—the DPO approach

Before formalizing our embedding of $\mathbb{K}$ rules into graph rules, we briefly recall some basic notions from the theory of graph grammars and graph transformations [2].

Assuming fixed sets $\mathcal{L}_V$ and $\mathcal{L}_E$ for node and for edge labels, respectively, a *graph G over labels* $(\mathcal{L}_V, \mathcal{L}_E)$ is a tuple $G = \langle V, E, \text{source}, \text{target}, \text{lv}, \text{le}\rangle$, where V is the set of *vertices* (or *nodes*), E is a set of *edges*, $\text{source}, \text{target} : E \to V$ are the *source* and the *target* functions, and $\text{lv} : V \to \mathcal{L}_V$ and $\text{le} : E \to \mathcal{L}_E$ are the node and the edge *labeling functions*, respectively. We will use V_G, E_G, source_G, ..., to refer to the corresponding components of the tuple describing a graph G. A *graph morphism* $f : G \to G'$ is a pair $f = \langle f_V : V_G \to V_{G'}, f_E : E_G \to E_{G'}\rangle$ of functions preserving sources, targets, and labels. Let $\mathbf{Graph}(\mathcal{L}_\mathbf{V}, \mathcal{L}_\mathbf{E})$ denote the category of graphs over labels $(\mathcal{L}_V, \mathcal{L}_E)$. Given graph G, let $\prec_G \subseteq V \times V$ be its *path relation*: $v_1 \prec_G v_2$ iff there is a path from v_1 to v_2 in G. G is *cyclic* iff there is some $v \in V_G$ s.t. $v \prec_G v$. Given $v \in V_G$, let $G{\restriction_v}$ be the subgraph of G (forwardly) reachable from v.

A *graph rewrite rule* $p : (L \xleftarrow{l} K \xrightarrow{r} R)$, is a pair of graph morphisms $l : K \to L$ and $r : K \to R$, where l is injective. The graphs L, K, and R are called the *left-hand-side* (lhs), the *interface*, and the *right-hand-side* (rhs) of p, respectively.

Given a graph G, a graph rule $p : (L \xleftarrow{l} K \xrightarrow{r} R)$, and a *match* $m : L \to G$, a *direct derivation from G to H using p (based on m)* exists iff the diagram to the right can be constructed, where both squares are pushouts in the category of graphs. In this case, C is called the *context* graph, and we write $G \overset{p,m}{\Longrightarrow} H$ or $G \overset{p}{\Rightarrow} H$. Whenever l or r is an inclusion, the corresponding l^* or r^* can be chosen to also be an inclusion.



A direct derivation $G \overset{p,m}{\Longrightarrow} H$ exists iff the following *gluing conditions* hold [2]: *(Dangling condition)* no edge in $E_G \setminus m_E(E_L)$ is incident to any node in $m_V(V_L \setminus l_V(V_K))$; and *(Identification condition)* there are no $x, y \in V_L \cup E_L$ with $x \neq y$, $m(x) = m(y)$ and $x, y \notin l(V_K \cup E_K)$. If it exists, H is unique up to graph isomorphism. The gluing conditions say that whenever a transformation deletes a node, it should also delete all its edges (dangling condition), and that a match is only allowed to identify elements coming from K (identification condition).

Given a family of graph-rewrite rules $p_i : (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$, $i = \overline{1,n}$, not necessarily distinct, their *composed graph-rewrite rule* $p_1 + \cdots + p_n$ is a rule $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ where L, K, and R are the direct sums of the corresponding components from $(p_i)_{i=\overline{1,n}}$ and, similarly, l and r are the canonical morphisms induced by $(l_i)_{i=\overline{1,n}}$ and $(r_i)_{i=\overline{1,n}}$, respectively. Given a graph G, matches $(m_i : L_i \to G)_{i=\overline{1,n}}$ induce a combined match $m : L \to G$ defined as the (unique) arrow amalgamating all individual matches. Matches $(m_i : L_i \to G)_{i=\overline{1,n}}$ have the *parallel independence* property iff for all $1 \leq i < j < n$, $m_i(L_i) \cap m_j(L_j) \subseteq m_i(K_i) \cap m_j(K_j)$. If $(m_i : L_i \to G)_{i=\overline{1,n}}$ have the parallel independence property and each m_i satisfies the gluing conditions for rule p_i, then the combined match m satisfies the gluing conditions for the composed rule $p_1 + \cdots + p_n$, and thus there exists a graph H such that $G \xRightarrow{p_1+\cdots+p_n,m} H$. Moreover, this derivation is serializable, i.e., $G \xRightarrow{p_1+\cdots+p_{n-1},m'} H_{n-1} \xRightarrow{p_n} H$, where m' is the composition of $(m_i)_{i=\overline{1,n-1}}$ [6, Theorem 7.3].

4 $\mathbb{K}$ Term-Graphs

$\mathbb{K}$ *term-graphs* are close to the bipartite graph representation of jungles (they actually coincide for ground terms). The difference is that the $\mathbb{K}$ term-graph representation allows certain variables (the anonymous and the pattern-hole variables) to be omitted from the graph. By reducing the number of nodes that need to be shared (i.e., by not forcing these variable nodes to be shared in the interface graph), this "partiality" allows terms at those positions to be concurrently rewritten by other rules.

The top-half of Figure 2 shows the $\mathbb{K}$ term-graphs involved in the graph representations of the $\mathbb{K}$ rules (1)–(4) of our running example. For example, the representation of variable x can be observed as the (singleton) graph R for rule (4), the constants a and b as graphs L and R from rule (3), and the term $f(x)$ as graph L in rule (4); all these $\mathbb{K}$ term-graphs are also graph jungles [5]. The bottom-half of Figure 2 shows the $\mathbb{K}$ term-graphs involved in the graph transformation which uses all four rules combined to rewrite the graph representation of $h(f(a), 0, 1)$ (graph G) to one that can be used to retrieve $h(g(b, b), 1, 0)$ (graph H).

The novel aspect of our representation is that, unlike the graph jungles, the $\mathbb{K}$ term-graphs are *partial*: they do not require each operation node to have outward edges for all sorts in its arity. This partiality plays a key role in "abstracting away" the anonymous variables and the holes of the pattern. For example, the number of outward edges specified for the nodes labeled with h have all possible values between 3 (its normal arity) in graphs G and H, to 0, e.g., in graph K for rule (1). This flexibility is crucial for enhancing concurrency; only through it rules (1) and (2) can apply in parallel, as it allows the outward edge of h labeled with 1 to be rewritten by rule (1), while h is still shared with rule (2). This is achieved by relaxing the properties of the graph representation of jungles to allow partially specified operations.

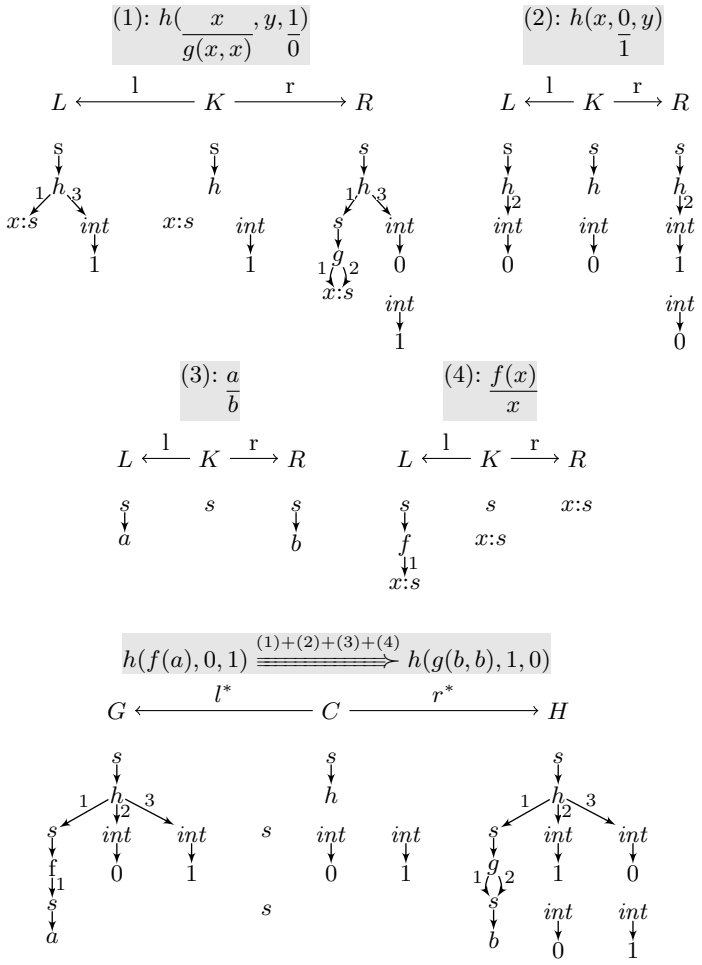


Fig. 2. Graph representations for the $\mathbb{K}$ rules (1)–(4) from the motivating example and their concurrent application.

Definition 2. *Given a signature $\Sigma = (S, F)$, a $\mathbb{K}$ Σ-**term-graph** is a graph G over labels $(S \cup F, \{\epsilon\} \cup Nat)$ satisfying:*

0. *G is bipartite, partitions given by nodes with labels in S—**sort nodes**—, and F—**operation nodes**—;*
1. *every operation node labeled by $f : s_1 \cdots s_n \to s$ is*
   (i) *the target of exactly one edge, labeled with 0 and having its source labeled with s, and*
   (ii) *the source of **at most** n edges having distinct labels in $\{1, \cdots, n\}$, such that $\mathrm{lv}(\mathrm{target}(e)) = s_{\mathrm{le}(e)}$ for each such edge e;*
2. *every sort node has at most one outward edge; and*
3. *G is acyclic.*

Let $\mathbf{KGraph}_{\mathbf{\Sigma}}$ *denote the full subcategory of* $\mathbf{Graph}(\mathbf{S} \cup \mathbf{F}, \{\epsilon\} \cup Nat)$ *having* $\mathbb{K}$ *Σ-term-graphs as objects.*

As any graph jungle is a $\mathbb{K}$ term-graph, most of the definitions from graph jungles [5] can be easily extended for term-graphs. For notational simplicity $\mathbb{K}$ term-graphs will be referred to as just term-graphs.

Given a set of anonymous variables $A \subseteq \mathcal{X}$, an A-anonymizing variable-collapsed tree representation of a term $t \notin A$ with variables from $\mathcal{X}$ is obtained from a variable-collapsed tree representing t (i.e., the tree representing t where variable nodes with same label have been identified) by removing the variable nodes corresponding to variables in A and their adjacent edges.

A *root* of term-graph is a (sort-)node v such that $\mathit{indegree}(v) = 0$. Let $ROOT_G$ denote the set of roots of G. VAR_G is the set variables of G, that is, sort-nodes of G such that $\mathit{outdegree}(v) = 0$. Note that this definition only captures the non-anonymous variables. To capture all variables, we need to additionally identify partially specified operation nodes.

Let G be a term-graph over $\Sigma = (S, F)$. The set $OPEN_G$ of *open (or incomplete) operation nodes* of G, consists of the operation nodes whose outward edges are incompletely specified. Formally, $OPEN_G = \{v \in \mathrm{lv}^{-1}(S) \mid |s^{-1}(v)| < \mathit{arity}(\mathrm{lv}(v))\}$. The set of *term variables* of G, $TVARS_G$ consists of the variables of G and the positions of the unspecified outward edges for open operation nodes (which stand for anonymous variables). Formally, $TVARS_G = VAR_G \cup \{x_{v,i} \mid v \in OPEN_G, 1 \leq i \leq \mathit{arity}(\mathrm{lv}(v)) \wedge i \notin \mathrm{le}(\mathrm{source}^{-1}(v))\}$.

The *term represented by some sort node v in a term-graph G*, $term_G(v)$, is obtained by descending along operation nodes and collecting their labels:

$$term_G(v_s) = \begin{cases} v_s, \text{ if } v_s \in VAR_G \\ \sigma(t_1, \ldots, t_n), \text{ if } \{v_e\} = \mathrm{target}(\mathrm{source}^{-1}(v_s)), \mathrm{le}(v_e) = \sigma : s_1 \ldots s_n \to s, \\ \qquad \text{and } t_i = subterm_G(v_e, i), 1 \leq i \leq n \end{cases}$$

where $subterm_G$ is defined on pairs of operation nodes with integers by

$$subterm_G(v_e, i) = \begin{cases} x_{v_e,i}, \text{ if } x_{v_e,i} \in TVARS_G \\ term_G(\mathrm{target}(e)), \text{ if } \mathrm{source}(e) = v_e \text{ and } \mathrm{le}(e) = i \end{cases}$$

5 $\mathbb{K}$ Graph Rewriting

As we want $\mathbb{K}$ graph rewriting to be a conservative extension of graph jungle evaluation, every 0-sharing $\mathbb{K}$ rule $(\forall \mathcal{X})\ \square[\ \frac{\mathit{left}}{\mathit{right}}\ ]$ is encoded as the graph jungle evaluation rule corresponding to the rewrite rule $\mathit{left} \to \mathit{right}$—see, for example the encodings of rules (3) and (4) in Figure 2. However, if the rule pattern k is non-empty, then the rule is encoded so that the variable-collapsed tree representing k would not be modified by the rule. To be more precise, instead of obtaining K by removing the outgoing edge from the root of L, we will instead only remove the edges connecting the hole variables to their parent operations.

Moreover, to further increase concurrency, the variables which appear in the read only pattern k but not in the left substitution are anonymized. However, departing from the definition of jungle rules, we relax the requirement that the order between the nodes of K and variables of R should be the same as in L, to allow rules such as reading or writing the value of a variable from a store.

Let us discuss the representation of the $\mathbb{K}$ rule (1) in Figure 2, namely $h(\underline{\quad x \quad}_{g(x,x)}, y, \underline{1}_{0})$. The left-hand-side is represented as a $\{y\}$-anonymized variable collapsed tree representing $h(x, y, 1)$; variable y is anonymized as only appearing in the pattern k. The interface K is obtained from L by severing (through the removal of edges labeled by 1 and 3) the part of L representing the read-only pattern $h(\square_1, y, \square_2)$ (which is the $\{y, \square_1, \square_2\}$-anonymized variable collapsed tree representing $h(\square_1, y, \square_2)$) from the parts of L representing the left substitution (namely, x and 1). Thus, the l morphism from K to L is clearly an inclusion. R is obtained by taking the disjoint union between K and the variable-collapsed trees corresponding to terms $g(x, x)$ and 0 given by the right substitution, identifying the variables, and "gluing" them to the part representing the read-only pattern through edges from operation node h labeled 1 and 3, respectively. Like l, the r morphism can also be chosen to be an inclusion.

The graph rules in Figure 2 are obtained using the definition below. To avoid clutter, we do not depict node or edge names (except for variables). Also, the actual morphisms are not drawn (they are either inclusions or obvious collapsing morphisms).

Definition 3. *Let $\rho : (\forall \mathcal{X})\ k[\ L \Rightarrow R\ ]$ be a $\mathbb{K}$ rule.*

If ρ is 0-sharing, then the $\mathbb{K}$ graph rewrite rules representing ρ coincide with the graph evaluation rules [5] corresponding to the rewrite rule associated to ρ.

Otherwise, a $\mathbb{K}$ ***graph rewrite rule*** *representing ρ is a graph rewrite rule $(L_\rho \xleftarrow{l_\rho} K_\rho \xrightarrow{r_\rho} R_\rho)$ such that:*

- L_ρ *is an A-anonymized variable collapsed tree representation of $L(k)$, where $A = vars(k) \setminus vars(L)$ are the anonymous variables of ρ;*
- K_ρ. *Let K_0 be the subgraph of L_ρ which is a A-anonymized variable collapsed tree representing k; then $K_\rho = (V_{K_\rho}, E_{K_\rho})$ is given by $V_{K_\rho} = V_{L_\rho}$ and $E_{K_\rho} = E_{L_\rho} \setminus \{e \in E_{L_\rho} \mid \text{source}(e) \in V_{K_0}$ and $\text{target}(e) \notin V_{K_0}\}$. l_ρ is the inclusion morphism.*
- R_ρ *Let R_0 be an A-anonymized variable collapsed tree representation of $R(k)$ containing K_0 as a subgraph. Then R_ρ is obtained as the pushout between the inclusions of $K_0 \cup VAR_{R_0}$ into K_ρ and R_0, respectively.*

The nodes from K_0 will be called pattern nodes.

Note that the edges removed from L_ρ to obtain K_ρ are those whose target corresponds to the hole variables of k.

Similarly to the graph jungle rules, the (basic) $\mathbb{K}$ graph rules defined above ensure that the gluing conditions are satisfied for any matching morphism. For the remainder of this section, let us fix G to be a term-graph, $\rho_i : (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$,

$i = \overline{1,n}$ to be $\mathbb{K}$ graph-rewrite rules, and $m_i : L_i \to G$ to be parallel independent matches. Let $\rho : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ be the composed rule of $(\rho_i)_{i=\overline{1,n}}$, and let $m : L \to G$ be the composition of the individual matches. It follows that m satisfies the gluing conditions for ρ, and thus (ρ, m) can be applied as a graph transformation. Let us now provide a concrete construction for the derivation of (ρ, m) in **Graph** which will be used in proving the subsequent results.

The pushout complement object of m and l can be defined in **Graph** as $C = G \setminus m(L \setminus K)$ where the difference is taken component-wise. That C is a graph is ensured by the gluing conditions. The standard construction of the pushout object H is to factor the disjoint union of C and R through the equivalence induced by the pushout morphism $\overline{m} : K \to C$ and r. We do this directly, by taking preference for elements in C, and thus choosing representatives from $\overline{m}(K)$ and by choosing as representatives variables for the equivalence classes induced by the parts of r belonging to collapsing rules.

Suppose G is a $\mathbb{K}$ graph representation of term t, i.e., that $ROOT_G = \{root_G\}$, $G = G\restriction_{root_G}$, and $term_G(root_G) = t$. When applying a (composed, or not) $\mathbb{K}$ graph rewrite rule to graph G, $root_G$ must be preserved in the context C, because K contains all nodes of L. Therefore, let us define the top of the obtained graph H as being $root_H = r^*(root_G)$. Note that $root_H$ might not be equal to $root_G$, because $root_G$ could be identified with a variable node by a collapsing rule; moreover, $root_H$ might not be the only element of $ROOT_H$, because of the potential "junk" left by the application of the rule. Nevertheless, the term $term_H(root_H)$ would be the one to which $term_G(root_G)$ was rewritten.

To show that $\mathbf{KGraph}_{\mathbf{\Sigma}}$ admits similar constructions for (composed) $\mathbb{K}$ graph-rewrite rules as **Graph**, that is, that the graphs described above are in fact term-graphs, we need to strengthen the constraints on the matching morphisms.

Indeed, without further constraints, applying $\mathbb{K}$ graph rules on term-graphs can produce cyclic graphs. Consider $\mathbb{K}$ rules $f(g(\underset{x}{a}), x)$ and $f(y, h(\underset{y}{b}))$ together with the term to rewrite $f(g(a), h(b))$. Upon formalizing terms as term-graphs and $\mathbb{K}$ rules as $\mathbb{K}$ graph rewrite rules, the result of applying the composed $\mathbb{K}$ graph rewrite rule on the graph representing $f(g(a), h(b))$ is the graph H in Figure 3(b), *which has a cycle* and thus it is not a term-graph.

The reason for the cycle being introduced is that the matches overlap, allowing variable nodes to precede operation nodes in the path order of G, while r reorders the mapping of the variables to create a cycle. In jungle rewriting [5] this issue is prevented by imposing a statically checkable condition on the rules, namely that the path relation between the nodes preserved from L should not be changed by R. Formally, we say that a rule $\rho : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ is *cycle free* if whenever $v \prec_R x$ with $v \in V_K$ and $x \in VAR_L \cap V_K$, it must be that $v \prec_L x$. This condition is sufficient to prevent the introduction of cycles; however, we find it rather strong in our programming language context—in particular, this condition would disallow rules like the one for reading the value of a variable from the store. In what follows, we give a (semantical) condition on the matching morphism m rather than the rule which is sufficient to avoid the introduction of cycles.

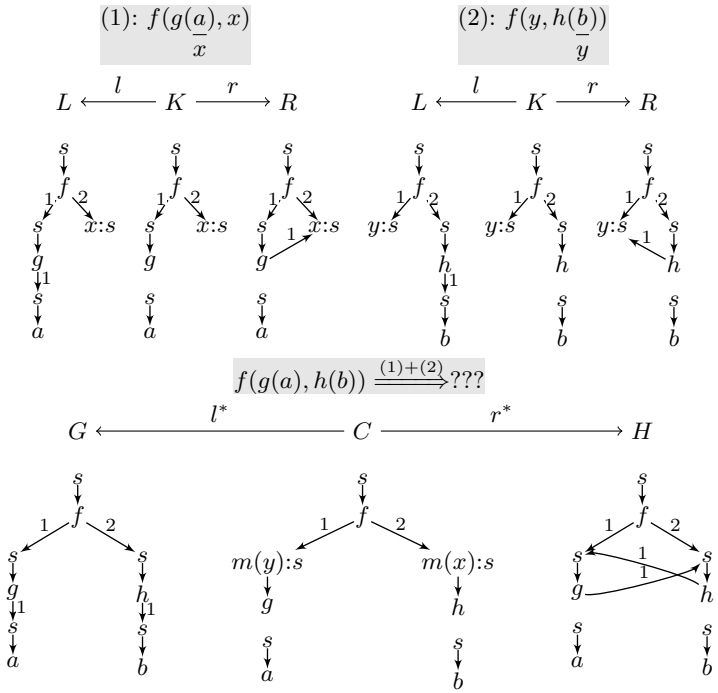


Fig. 3. Parallel $\mathbb{K}$ graph rewriting can introduce cycles.

Given a (composed) term-graph rewrite rule $\rho : (L \xleftarrow{l} K \xrightarrow{r} R)$, r induces on K a (partial) replacement order $\prec_r = r^{-1}(\prec_R)$, i.e., $v_1 \prec_r v_2$ in K iff $r(v_1) \prec_R r(v_2)$ (there is a path from $r(v_1)$ to $r(v_2)$ in R). Moreover, given match m of p into G, m induces on K a (partial) matching order $\prec_m = l^{-1}(m^{-1}(\prec_G))$, i.e., $v_1 \prec_r v_2$ in K iff $m(v_1) \prec_G m(v_1)$ (l is an inclusion). Although both these (partial) orders are strict, their combination is not guaranteed to remain strict. We say that the match m is *cycle free* w.r.t. p if the transitive closure of $\prec_m \cup \prec_r$ is also a strict (partial) order.

Proposition 1. *If any matching morphism for a $\mathbb{K}$ graph rewriting rule ρ is cycle free, then ρ is a jungle graph rewriting rule. If ρ is a $\mathbb{K}$ graph rule, G is a term-graph, $G \overset{(\rho,m)}{\Longrightarrow} H$, and m is cycle free w.r.t. ρ, then H is acyclic.*

The following result guarantees that if the original graph is a tree, then cycle freeness of the matching morphism characterizes acyclicity of the resulting graph.

Proposition 2. *Let G be a tree term-graph. If ρ is a simple $\mathbb{K}$ graph rule and m is a match for ρ into G, then m is cycle free. If ρ is a composed $\mathbb{K}$ graph rule and $G \overset{(\rho,m)}{\Longrightarrow} H$, then H is acyclic iff m is cycle free w.r.t. ρ.*

Next result shows that, under cycle-freeness conditions, $\mathbf{KGraph}_{\mathbf{\Sigma}}$ is closed under (parallel) derivations using $\mathbb{K}$ graph rewrite rules.

Theorem 1. *Let G, $(\rho_i)_{i=\overline{1,n}}$, $(m_i)_{i=\overline{1,n}}$, ρ, m, C, and H be defined as above. If m is cycle-free w.r.t. p, then:*

(Parallel) Derivation: $G \xRightarrow[\mathbf{KGraph_\Sigma}]{\rho,m} H$;

Serialization: *There exist $(G_i)_{i=\overline{0,n}}$ such that $G_0 = G$, $G_n = H$, and $G_{i-1} \xRightarrow[\mathbf{KGraph_\Sigma}]{\rho_i} G_i$ for each $1 \leq i \leq n$.*

6 $\mathbb{K}$ Term Rewriting

Theorem 1 allows us to capture the serializable fragment of $\mathbb{K}$ rewriting as the relation $\Rrightarrow$ defined below:

Definition 4. *Let t be a Σ-term and $\rho_1, \cdots, \rho_n$ be $\mathbb{K}$ rules (not necessarily distinct). Then $t \overset{\rho_1+\cdots+\rho_n}{\Longrightarrow} t'$ iff there is a term-graph H s.t. $G \xRightarrow[\mathbf{KGraph_\Sigma}]{K2G(\rho_1)+\cdots+K2G(\rho_n)} H$ and $term_H(\top_H) = t'$, where G is the tree term-graph representing t. We say that $t \Rrightarrow t'$ iff there is a (composed) $\mathbb{K}$ rule ρ s.t. $t \overset{\rho}{\Rrightarrow} t'$.*

We next show that the $\mathbb{K}$ rewriting above is a conservative extension of the standard term rewriting relation.

We can give a straightforward definition for what it means for a $\mathbb{K}$ rule to match a term: *one* $\mathbb{K}$ rule $\rho : (\forall \mathcal{X})\ k[\ L \Rightarrow R\ ]$ matches a term t using context C and substitution θ iff its corresponding rewrite rule $K2R(\rho) : (\forall X) L(k) \to R(k)$ matches t using the same C and θ, that is, iff $t = C[\theta(L(k))]$. This conforms to the intuition that, when applied sequentially, $\mathbb{K}$ rules behave exactly as their corresponding rewrite rules.

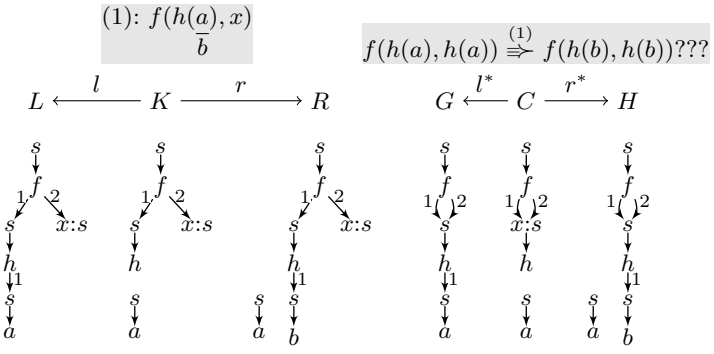


Fig. 4. Subterm sharing might lead to unsound $\mathbb{K}$ graph rewriting.

However, it turns out that, although preserving the term-graph structure (under cycle-freeness assumptions), $\mathbb{K}$ rewriting on graphs might not be sound w.r.t. term rewriting in the presence of subterm sharing. Consider the example in Figure 4. We want to apply rule $f(h(a), x)$, corresponding to the regular rewrite $\overline{b}$

rule $f(h(a), x) \to f(h(b), x)$, to the term $f(h(a), h(a))$. If we would represent $f(h(a), h(a))$ as a tree, then the $\mathbb{K}$ graph rewriting step would be sound, leading to a graph depicting $f(h(b), h(a))$; however, if we decide to collapse the tree representing $h(a)$ then we obtain $f(h(b), h(b))$, as depicted in Figure 4 which cannot be obtained through regular rewriting. The reason for this unsound rewriting is that part of the read-only patten of the rule is shared. To overcome this, we will restrict the read-only pattern of the rule to only match against a tree in the graph to be rewritten. We say that a match $m : L \to G$ of a $\mathbb{K}$ graph rewrite rule $\rho : (L \xleftarrow{l} K \xrightarrow{r} R)$ is *safe* if $m(K\restriction_{root_L})$ is a tree in G, that is, if $indegree_G(m_V(v)) = 1$ for any $v \in V_{K\restriction_{root_L}} \setminus \{root_L\}$. Note that, if G is a tree then all matching morphisms on G are safe.

Proposition 3. *Let ρ be a proper $\mathbb{K}$ rewrite rule, let ρ_0 be its associated 0-sharing $\mathbb{K}$ rewrite rule, and let m be a cycle free safe matching morphism for $K2G(\rho)$ in G. Let H be such that $G \xRightarrow[\mathbf{KGraph_\Sigma}]{K2G(\rho),m} H$, and let H' be such that $G \xRightarrow[\mathbf{KGraph_\Sigma}]{K2G(\rho_0),m} H'$. Then for any $v \in ROOT_G$, $term_H(v) = term_H(v)$.*

Since the $\mathbb{K}$ graph representation of a term t without anonymous variables is a graph jungle representing the same term, and since the $\mathbb{K}$ term-graph representation of a 0-sharing $\mathbb{K}$ rewrite rule is a graph jungle rule representing the rewrite rule associated to it, we can use the soundness and completeness of jungle rewriting w.r.t. standard term rewriting [5] to prove the sequential soundness and completeness of $\mathbb{K}$ graph rewriting w.r.t. standard term rewriting, and, by combining that with Theorem 1, to prove the serializability result for $\mathbb{K}$ rewriting.

Theorem 2. *Let ρ, ρ_1, ..., ρ_n be $\mathbb{K}$ rules. Then:*

Completeness: *If $t \xRightarrow{K2R(\rho)} t'$ then $t \overset{\rho}{\Rightarrow} t'$.*

Soundness: *If $t \overset{\rho}{\Rightarrow} t'$ then $t \xRightarrow{K2R(\rho)^*} t'$.*

Serializability: *If $t \xRightarrow{\rho_1+\cdots+\rho_n} t'$, then there exists a sequence of terms $t_0, \cdots, t_n$, such that $t_0 = t$, $t_n = t'$, and $t_{i-1} \overset{\rho_i^*}{\Rightarrow} t_i$.*

Therefore, $\mathbb{K}$ rewriting is sound and complete for term rewriting, while providing a higher degree of concurrency in one step than existing approaches.

7 Conclusion

This paper presents a truly concurrent semantics with sharing of resources for the $\mathbb{K}$ framework, a term-rewriting-based semantic framework specialized for defining programming languages and calculi. The distinguishing aspect of the $\mathbb{K}$ rewrite rules is that they explicitly state what portions of the term can be concurrently shared with other rules. This sharing information allows one to increase the potential for concurrent rewriting, but it may also lead to inconsistencies if not used properly. We showed that, under reasonable conditions, $\mathbb{K}$ rewriting is actually sound, complete, and serializable w.r.t. term rewriting. Moreover,

although being motivated by the $\mathbb{K}$ framework, $\mathbb{K}$ rewriting is not confined to it; it rather is an extension of rewriting which allows additional concurrency for any rewriting-based formalism.

Future work Although we have found a sufficient condition for sound and serializable concurrent executions, this condition is rather semantical, and might be non-trivial to check. However, all of the rule combinations in our current definitions of programming languages seem to generate cycle free executions. An interesting research problem would be to find generic enough syntactic conditions which would guarantee that cycle freeness is satisfied for all possible combinations of matches.

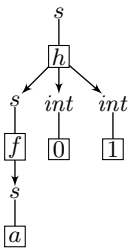
References

1. G. Berry and G. Boudol. The chemical abstract machine. *J. of Theoretical Computer Science*, 96(1):217–248, 1992.
2. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation: Basic concepts and double pushout approach. In *Handbook of graph grammars*, volume 1, pages 163–246. World Sci., 1997.
3. C. Ellison and G. Roşu. An executable formal semantics of C with applications. In *POPL*, pages 533–544, 2012.
4. M. Felleisen and D. P. Friedman. Control operators, the SECD-machine, and the lambda-calculus. In *3rd Working Conference on the Formal Description of Programming Concepts*, pages 193–219, Ebberup, Denmark, Aug. 1986.
5. A. Habel, H.-J. Kreowski, and D. Plump. Jungle evaluation. In *ADT*, volume 332 of *LNCS*, pages 92–112, 1987.
6. A. Habel, J. Müller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
7. P. Meredith, M. Hills, and G. Roşu. An executable rewriting logic semantics of K-Scheme. In *SCHEME*. Technical Report DIUL-RT-0701, pages 91–103. Laval University, 2007.
8. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *J. of Theoretical Computer Science*, 96(1):73–155, 1992.
9. J. Meseguer. Rewriting logic as a semantic framework for concurrency. In *CONCUR*, volume 1119 of *LNCS*, pages 331–372, 1996.
10. U. Montanari. True concurrency: Theory and practice. In *MPC*, volume 669 of *LNCS*, pages 14–17, 1992.
11. D. Plump. Term graph rewriting. In *Handbook of graph grammars*, volume 2, pages 3–61. World Sci., 1999.
12. G. Roşu and T. F. Şerbănuţă. An overview of the K semantic framework. *J. of Logic and Algebraic Programming*, 79(6):397–434, 2010.
13. T. F. Şerbănuţă. *A Rewriting Approach to Concurrent Programming Language Design and Semantics*. PhD thesis, University of Illinois, December 2010.
14. T. F. Şerbănuţă and G. Roşu. KRAM—extended report. Technical Report http://hdl.handle.net/2142/17337, UIUC, September 2010.
15. A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.

The material presented below is included for reviewers' convenience and can be found in the companion technical report [14].

A Jungle rewriting

The jungle term-graph rewriting [5] approach uses (directed) hypergraphs to encode terms and rules. Hypergraphs generalize graphs by allowing edges to have multiple (or zero) sources and targets; more precisely, the source and target mappings now yield words over nodes instead of just a node (words, rather than sets, to maintain an order among nodes). A jungle represents a term as an acyclic hypergraph whose nodes are labeled by sort names, and whose edges are labeled by names of operations in the signature; the above figure on the right depicts the jungle representation of term $h(f(a), 0, 1)$. Constants are edges without any target. Variables are represented as nodes which are not sources of any edge. Non-linear terms are represented by identifying the nodes corresponding to the same variable. There could be multiple possible representations of the same term as a jungle, as identical subterms can be identified (or not) in the jungle representation.



Let VAR_G denote the *variables of* G; we have that $VAR_G = \{v \in V_G \mid outdegree_G(v) = 0\}$. The *term represented by some node* v *in a jungle* G, $term_G(v)$, is obtained by descending along hyperedges and collecting the hyperedge labels:

$$term_G(v) = \begin{cases} v \text{ if } v \in VAR_G \\ \mathrm{le}(e)(term^*_G(\mathrm{target}(e)) \text{ otherwise,} \end{cases} \quad \text{where } \{e\} = \mathrm{source}^{-1}(v).$$

A *root* of a jungle is a node v such that $indegree(v) = 0$. Let $ROOT_G$ denote the set of roots of G. Given a term t (with variables), a *variable-collapsed* tree representing t is a jungle G with a single root $root_G$ which is obtained from the tree representing t by identifying all nodes corresponding to the same variable, that is, $term_G(root_G) = t$, and for all $v \in V$, $indegree(v) > 1$ implies that $v \in VAR_G$.

A term rewrite rule $left \to right$ is encoded as a *jungle evaluation rule* $L \hookleftarrow K \xrightarrow{r} R$ in the following way: L is a variable-collapsed tree corresponding to *left*; K is obtained from L by removing the hyperedge corresponding to the top operation of *left*; if *right* is a variable (i.e., the rule is collapsing, then R is obtained from K by identifying $root_L$ with *right*; otherwise, R is the disjoint union of K and a variable collapsing tree R' corresponding to *right*, where $root_{R'}$ is identified with $root_L$ and each variable of R' is identified with its counterpart from VAR_L; $L \xhookleftarrow{l} K$ and $K \xrightarrow{r} R$ are inclusions with the exception that r maps $root_L$ to *right* if *right* is a variable.

Since each edge has precisely one source, it is easy to reconstruct a term given a jungle and specifying a top node. It has been shown [5] show that graph rewriting on jungles is sound and complete for term rewriting.

By using the bipartite graph model of hypergraphs, jungle hypergraphs can be represented as bipartite graphs, where both sort nodes and operation edges of

the hypergraph become sort and operation nodes in the graph and new edges are added to link them similarly to how they were linked in the hypergraph. To maintain the order given by the target word, we here prefer to label the corresponding edges with their position in that word. For example, the bipartite graph of the jungle representing $h(f(a), 0, 1)$ is represented by graph G in Figure 2.

B Proofs of the results

Let us first state some facts given by the structure of $\mathbb{K}$ graph rewrite rules. Let $root_i$ identify the root of L_i in L. Since the lhs cannot be a variable, it follows that L_i has at least one edge and one operation node. K_i is a subgraph of L_i and l_i is the inclusion morphism; moreover K_i contains all nodes of L_i.

We have that $ROOT_L = \{root_i \mid i = \overline{1,n}\}$. Let now J be the set of indexes of collapsing rules. In the following, let i range over $\{1, \ldots, n\}$ and let j range over J.

We define H, together with $r^* : C \to H$ and $m^* : R \to H$, as follows:

- $V_H = (V_C \setminus \{m(root_j) \mid j \in J\}) \uplus (V_R \setminus V_K)$
- $r^*_V(v) = \begin{cases} v, & \text{if } v \neq m(root_j), \\ r^*_V(m(r(root_j))), & \text{if } v = m(root_j), \end{cases}$
- $m^*_V(v) = \begin{cases} v, & \text{if } v \notin V_K \\ r^*_V(m_V(v)), & \text{otherwise} \end{cases}$
- $E_H = E_C \uplus (E_R \setminus E_K)$
- $r^*_E(e) = e$ and $m^*_E(e) = \begin{cases} e, & \text{if } e \notin E_K \\ m_E(e), & \text{otherwise} \end{cases}$
- $\text{source}_H(e) = \begin{cases} r^*(\text{source}_C(e)), & \text{if } e \in E_C \\ m^*_V(\text{source}_R(e)), & \text{if } e \in E_R \setminus E_K \end{cases}$
- $\text{target}_H(e) = \begin{cases} r^*_V(\text{target}_C(e)), & \text{if } e \in E_C \\ m^*_V(\text{target}_R(e)), & \text{if } e \in E_R \setminus E_K \end{cases}$

Note that r^*_V is recursively defined. However, it is well defined, because G is acyclic and, since $r_V(root_j) \in VAR_{L_j}$, it must be that $G{\restriction}_{m_V(r_V(root_j))}$ is a strict subgraph of $G{\restriction}_{m_V(root_j)}$, implying that the recursion should end because both G and J are finite. It can be easily verified that (H, r^*, m^*) is a pushout of $(\overline{m}, r)$.

Proposition 1. *(1) If any matching morphism for a $\mathbb{K}$ graph rewriting rule ρ is cycle free, then ρ is a jungle graph rewriting rule. (2) If ρ is a $\mathbb{K}$ graph rule, G is a term-graph, $G \overset{(\rho,m)}{\Longrightarrow} H$, and m is cycle free w.r.t. ρ, then H is acyclic.*

Proof. Let $\rho : (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ be a $\mathbb{K}$ graph rewriting rule.

(1) Suppose that there exist $v \in V_K$ and $x \in VAR_L$ such that $v \prec_R x$ and $v \not\prec_L x$. Let then G be the graph obtained from L by adding an edge e such that $\text{source}(e) = x$ and $\text{target}(e) = v$. G is still acyclic, because L is acyclic and because $v \not\prec_L x$. Let $m : L \to G$ be the inclusion morphism. We have that m is not cycle free, since $v \prec_R x$ implies that $v \prec_r x$ and $x \prec_G v$ implies that $x \prec_m v$, contradiction.

(2) Proof by contradiction. Assume that H is not acyclic, and let $e_0, \ldots, e_n$ be a sequence of edges in H exhibiting a cycle. Let then $e_{\alpha_0}, \ldots, e_{\alpha_m}$ be a subsequence of the above sequence with the property that all its elements are edges in C and that the blocks of edges between them (including the one starting at e_{α_m} and wrapping over to e_{α_0} are alternating between C and $R \setminus K$. Then, if the edges between e_{α_i} and $e_{\alpha_{i+1}}$ are all in C, it must be that both source(e_{α_i}) and target$(e_{\alpha_{i+1}})$ are in V_K, and moreover, that source$(e_{\alpha_i}) \prec_m$ target$(e_{\alpha_{i+1}})$. Similarly, if the edges between e_{α_i} and $e_{\alpha_{i+1}}$ are in $R \setminus K$, then both target(e_{α_i}) and source$(e_{\alpha_{i+1}})$ are in V_K (which we already knew from the previous sentence) and that target$(e_{\alpha_i}) \prec_r$ source$(e_{\alpha_{i+1}})$. But this precisely implies that m is not cycle free, contradiction.

Proposition 2. *Let G be a tree term-graph.*

1. *If ρ is a simple $\mathbb{K}$ graph rule and m is a match for ρ into G, then m is cycle free.*
2. *If ρ is a composed $\mathbb{K}$ graph rule and $G \overset{(\rho,m)}{\Longrightarrow} H$, then H is acyclic iff m is cycle free w.r.t. ρ.*

Proof. Observation 1: Since G is a tree, $v_1 \prec_G v$ and $v_2 \prec_G v$ implies that either $v_1 \prec_G v_2$ or $v_2 \prec_G v_1$.

Observation 2: Assuming m is not cycle free, since both $\prec_m$ and $\prec_r$ are acyclic, it must be that the cycle is obtained by an alternating sequence $v_1 \prec_r x_1 \prec_m v_2 \ldots \prec_r x_{n-1} \prec_m v_n = v_1$, where x_i is a variable node and v_i is a pattern node for all $1 \leq i < n$.

(1) Let us show that is impossible to have $x \prec_m v$ where x is a variable node and v is a pattern node, whence m must be cycle free. Indeed, $x \prec_m v$ means that $m(x) \prec_G m(v)$, which would lead to $x \prec_L v$ (since $m(L)$ is a subtree of G), which is not possible, as x is a leaf in L.

(2) We only need to prove that if m is not cycle-free, then H has cycles, as the converse was proven in the general case by Proposition 1. Assume m is not cycle free, and consider a minimal sequence exhibiting a cycle as in Observation 2. We want to show that this sequence is also valid if we replace $\prec_m$ with $\prec_{\overline{m}} = \overline{m}^{-1}(\prec_C)$, which would necessarily lead to a cycle in H, as H is obtained as the pushout between C and R identifying K. We again reason by contradiction and assume that this is not the case, that is, there exists $1 \leq i < n$ such that $x_i \prec_m v_{i+1}$ but $x_i \not\prec_{\overline{m}} v_{i+1}$. However, this can only happen if an edge between $m(x_i)$ and $m(v_{i+1})$ in G is removed by another rule. Therefore, there must exist a pattern node v and a variable node x such that $x_i \prec_m v$, $v \prec_L x$, $x \prec_m v_{i+1}$, and $v \not\prec_K x$. From $v_{i+1} \prec_r x_{i+1}$ we deduce that $v_{i+1} \prec_r x_{i+1}$ are part of the same rule, and therefore there must be some $v' \in K$ such that $v' \prec_L v_{i+1}$ and $v' \prec_L x_{i+1}$. Using Observation 1, $m(v) \prec_G m(x) \prec_G m(v_{i+1})$ and $m(v') \prec_G m(v_{i+1})$ implies that either $m(v) \prec_G m(v')$ or $m(v') \prec_G m(v)$. Using the parallel independence condition we deduce that $m(v) \prec_G m(v')$, whence $x_i \prec_m v \prec_m v' \prec_m x_{i+1} \prec_m v_{i+2}$. However, $x_i \prec_m v_{i+2}$ is in contradiction with our original assumption that the cycle was minimal.

Theorem 1. *Let G, $(\rho_i)_{i=\overline{1,n}}$, $(m_i)_{i=\overline{1,n}}$, ρ, m, C, and H be defined as above. If m is cycle-free w.r.t. p, then:*

(Parallel) Derivation: $G \xRightarrow[\mathbf{KGraph_\Sigma}]{\rho,m} H$*;*

Serialization: *There exist $(G_i)_{i=\overline{0,n}}$ such that $G_0 = G$, $G_n = H$, and $G_{i-1} \xRightarrow[\mathbf{KGraph_\Sigma}]{\rho_i} G_i$ for each $1 \leq i \leq n$.*

Proof. From the parallel independence condition, there exists a derivation $G \xRightarrow{\rho,m} H$ in **Graph**, and, H must be acyclic (Proposition 1). To prove the Derivation claim we only need to show that the graphs produced by the derivation, C and H, are indeed term-graphs.

Assuming that we have proved the Derivation claim, we can use the serializability result for the category of graphs iteratively, the first step being the following: From $G \xRightarrow{p_1+\cdots+p_n,m} H$ we deduce that $G \xRightarrow{p_1+\cdots+p_{n-1},m'} H' \xRightarrow{p_n} H$, where m' is the composition of $(m_i)_{i=\overline{1,n-1}}$; however, by the derivation claim, H' is also a term-graph, and, therefore, we can iterate to obtain the serialization result in $\mathbf{KGraph_\Sigma}$.

To prove the derivation part of the theorem, we only need to show that the graphs C and H defined above are term-graphs. First, let us show that C is a term-graph. Conditions (0)—C is bipartite, (1.ii) at most n consistently labeled outward edges for each operation node, (2)—at most one outward edge for each sort node, and (3)—C is acyclic are obviously satisfied, since we only remove nodes and edges. For (1.i) we only need to notice that whenever $e \in E_L \setminus E_K$ such that source(e) is a sort node then target$(e) \in V_L \setminus V_K$ since it is the root operation node corresponding to a 0-sharing rule. Let $l^* : C \to G$ and $\overline{m} : K \to C$ be the morphisms completing the pushout diagram. We have that l^* is an inclusion and $\overline{m}$ is the restriction and co-restriction of m to K and C, respectively.

Let us now additionally verify that H is a term-graph.

(0)—H is bipartite This is ensured by the fact that R is bipartite and r only identifies nodes of the same kind.

(1.i)—each operation node has exactly one inward edge Proof by contradiction. Suppose there exists distinct edges e, e' in E_H such that $\text{target}_H(e) = \text{target}_H(e)$ and it is an operation node. Since $\top_i$ and $r_V(\top_i)$ are sort nodes, we can assume, as above that $e \in E_C$, $e' \in E_R \setminus E_K$, $\text{target}_R(e') \in V_K$, and $\text{target}_C(e) = m_V(\text{target}_R(e'))$. However, $e' \in E_R \setminus E_K$, $\text{target}_R(e') \in V_K$, and $\text{target}_R(e')$ operation node constitute a contradiction with the fact that R satisfies $(1.i)$, since there should be another edge in E_K with the same target as e'.

(1.ii)—each operation node's outward edges are consistent Since both C and R are term-graphs, the labels of outward edges of operation sorts, as well as the labels of their targets must be consistent in H. To complete our proof we only need to additionally show that no duplicates are introduced by the merging. Proof by contradiction. Suppose there exists distinct edges e and e' in E_H, such

that $\text{source}_H(e) = \text{source}_H(e')$ is an operation node, and $\text{le}_H(e) = \text{le}_H(e')$. Then we can assume that $e' \in E_C$, $e \in E_R \setminus E_K$, and $\text{source}_R(e) \in V_K$, inducing that $\text{source}_C(e') = m_V(\text{source}_R(e))$. From $e \in E_R \setminus E_K$ and $\text{source}_R(e) \in V_K$ we infer that there exists i such that $e \in E_{R_i} \setminus E_{K_i}$ and $s_{R_i}(e) \in V_K$ is an operation node. Therefore, $x_{\text{source}_{R_i}(e),\text{le}_{R_i}(e)}$ cannot be a (term) variable of R_i, and therefore, it cannot be a term variable of L_i, as well. Moreover, since $\text{source}_{R_i}(e) \in V_K$, it must be that $\text{source}_{R_i}(e) \in V_{L_i}$, and hence there exists $e_i \in E_{L_i}$ such that $\text{source}_{L_i}(e_i) = \text{source}_{R_i}(e)$ and $\text{le}_{L_i}(e_i) = \text{le}_{R_i}(e)$. But this implies that $e_i \in E_{L_i} \setminus E_{K_i}$, which contradicts with the fact that $e' \in E_C$ (since e' has the same source and label).

(2)—each sort node has at most one outward edge Proof by contradiction. Suppose there exist distinct edges e and e' in E_H such that $\text{source}_H(e) = \text{source}_H(e') = v$, and v is a sort node. We can then suppose (without loss of generality) that $e \in E_C$ and $e' \in E_R \setminus E_K$. Then $\text{source}_R(e') \in V_K$ and $v = \text{source}_H(e) = \text{source}_C(E) = m_V(\text{source}_R(e'))$. Reusing a previous argument, from $\text{source}_r(e') \in V_K$, $e' \in E_R \setminus E_K$ and $\text{source}_R(e')$ sort node we deduce that $\text{source}_R(r') \in ROOT_L$. Therefore, there exists i such that $e' \in E_{R_i} \setminus E_{K_i}$ and $\text{source}_{R_i}(e') = \top_i$. However, this implies that $\text{source}_C^{-1}(m_V(\text{source}_{R_i}(e'))) = \emptyset$, which contradicts with $e \in E_C$.

(3)—H is acyclic This is ensured by the hypothesis that m is cycle-free w.r.t. p.

Proposition 3. *Let ρ be a proper $\mathbb{K}$ rewrite rule, let ρ_0 be its associated 0-sharing $\mathbb{K}$ rewrite rule, and let m be a cycle free safe matching morphism for $K2G(\rho)$ in G. Let H be such that $G \xRightarrow[\mathbf{KGraph_\Sigma}]{K2G(\rho),m} H$, and let H' be such that $G \xRightarrow[\mathbf{KGraph_\Sigma}]{K2G(\rho_0),m} H'$. Then for any $v \in ROOT_G$, $term_H(v) = term_H(v)$.*

Proof. First, cycle freeness ensures the existence of H; moreover, any 0-sharing $\mathbb{K}$ rule generates the graph representation of a jungle evaluation rule, and thus the existence of H' is ensured.

Second, since ρ is proper, neither ρ nor ρ_0 is collapsing, and therefore $ROOT_G \subseteq ROOT_H$ and $ROOT_G \subseteq ROOT_{H'}$, so the final claim is also defined.

Let $K2G(\rho) : (L_\rho \xleftarrow{l_\rho} K_\rho \xrightarrow{r_\rho} R_\rho)$ and $K2G(\rho_0) : (L_\rho \xleftarrow{l_{\rho_0}} K_{\rho_0} \xrightarrow{r_{\rho_0}} R_{\rho_0})$ be the complete descriptions of $K2G(\rho)$ and $K2G(\rho_0)$, and let C, C' be the corresponding context graphs obtained in the process of applying the rules to G.

We have that $C = G \setminus m(L_\rho \setminus K_\rho)$, whence $V_C = V_G$ and $E_C = E_G \setminus \{m(e_{\Box_i}) \mid e_{\Box_i} \in E_{L_\rho}, \text{target}(e_{\Box_i}) \text{ corresponds to } \Box_i \in \mathcal{W}\}$. Also $C' = G \setminus m(L_\rho \setminus K_{\rho_0})$, whence $V_{C'} = V_G \setminus m_V(v_0)$ and $E_{C'} = E_G \setminus (\text{source}^{-1}(v_0) \cup \text{target}^{-1}(v_0))$, where $v_0 = \text{target}(\text{source}^{-1}(root_{L_\rho}))$.

H' is obtained by "gluing" on C' $R_{\rho_0} \setminus K_{\rho_0}$, that is the variable collapsed tree representation of $R(k)$ in which the root and the variable nodes have been removed. This gluing is done by setting the source of the topmost edge to be $m_V(root_L)$ and the target of any edge whose target is variable node x in R_{ρ_0} to be $m_V(x)$.

H is obtained by "gluing" on C $R_\rho \setminus K_\rho$, that is the variable collapsed tree representation of $\{R(Hole) \mid \Box \in \mathcal{W}\}$ in which the variable nodes have been removed, and an edge $e'_{\Box_i}$ for each $\Box_i$ has been added having as target the node representing the root of $R(\Box_i)$. The gluing is done by setting the target of any edge whose target is variable node x in R_ρ to be $m_V(x)$ and by setting the source of $e'_{\Box_i}$ to be $\text{source}_G(e_{\Box_i})$.

We can define a morphism $f : H \to H'$, as follows: For $C \setminus m(K_0)$ it is the identity: $f_V(v) = v if v \in V_C \setminus m(V_{L_\rho}) = V_G \setminus m(V_{L_\rho})$. $f_E(e) = e if e \in E_C \setminus m(E_{L_\rho}) = E_G \setminus m(E_{L_\rho})$. For the root of L_ρ and for its variables, it is also the identity: $f_V(m(root_{L_\rho})) = m(root_{L_\rho})$; $f_V(m(x)) = m(x)$. Now, for $K_0 = K_\rho{\restriction}_{root_{L_\rho}}$, it yields the copy of K_0 in R: $f_V(m_V(v)) = v$ for any $v \in V_{K_\rho{\restriction}_{root_{L_\rho}}}$, $v \neq root_{L_\rho}$ and $f_V(m_V(e)) = e$ for any $e \in E_{K_\rho{\restriction}_{root_{L_\rho}}}$. Finally, the mapping $R_\rho \setminus K_\rho$ it is already determined by the mapping of the elements coming from K_0, ad basically says that the variable collapsed trees corresponding to $R(\Box_i)$ are mapped to their corresponding (variable collapsed) subtrees coming from R_{ρ_0}.

It is relatively easy to verify that f is an injective morphism. Moreover the nodes and edges which are not in its image are part of the graph $m(K_0)$ (excluding $root_L$ and the topmost operation node as well as its adjacent edges), which, by being required to be a tree in G, has no incoming edge, and thus is not part of $\overline{H'} = H'{\restriction}_{ROOT_G}$. Hence, the restriction and co-restriction of f to $\overline{H} = H{\restriction}_{ROOT_G}$ and $\overline{H'}$, respectively, is a bijection, and, therefore for any $v \in ROOT_G$, $H{\restriction}_v$ is isomorphic with $H'{\restriction}_v$, whence $term_H(v) = term_{H'}(v)$.

Theorem 2. *Let ρ, ρ_1, ..., ρ_n be $\mathbb{K}$ rules. Then:*

Completeness: *If $t \xRightarrow{K2R(\rho)} t'$ then $t \overset{\rho}{\Rightarrow} t'$.*

Soundness: *If $t \overset{\rho}{\Rightarrow} t'$ then $t \xRightarrow{K2R(\rho)^*} t'$.*

Serializability: *If $t \xRightarrow{\rho_1+\cdots+\rho_n} t'$, then there exists a sequence of terms $t_0, \cdots, t_n$, such that $t_0 = t$, $t_n = t'$, and $t_{i-1} \overset{\rho_i^*}{\Rightarrow} t_i$.*

Proof. Let G be the tree term-graph representation of t.

Completeness From the completeness of jungle evaluation, we infer that there exists H such that $G \xRightarrow{m,K2G(\rho_0)} H$ and $term_H(m^*(root_G)) = t'$. Since G is a tree, m must be both cycle-free and safe for $K2G(\rho)$. From Proposition 3 we then infer that $G \xRightarrow{m,K2G(\rho)} H'$ and that $term_{H'}(m^*(root_G)) = t'$, whence $t \overset{\rho}{\Rightarrow} t'$.

Soundness From $t \overset{\rho}{\Rightarrow} t'$ it follows that there exists a rewrite sequence $G \xRightarrow{m,K2G(\rho)} H'$ such that $term_{H'}(m^*(root_G)) = t'$. Again, since G is a tree, m must be both cycle free and safe, whence, by Proposition 3, $G \xRightarrow{m,K2G(\rho_0)} H$ such that $term_H(m^*(root_G)) = t'$, and by the soundness of jungle evaluation, $t \xRightarrow{K2R(\rho)^*} t'$.

Serializability $t \overset{\rho_1+\cdots+\rho_n}{\Longrightarrow} t'$ implies that $G \overset{K2G(\rho_1)+\ldots+K2G(\rho_n)}{\Longrightarrow} H$ such that $term_H(m^*(root_G)) = t'$. Applying Theorem 1, we deduce that there exist $G_0, G_1, \ldots, G_n$ such that $G_0 = G$, $G_n = H$, and $G_{i-1} K2G(\rho_1) G_i$. Since G is a tree and all rules satisfy the parallel independence property, we can deduce that the matching morphism for each of the steps is safe, and thus also cycle free. Therefore we can for each step apply Proposition 3, and the the soundness of jungle evaluation w.r.t. rewriting, to obtain the desired answer.