Greedily Decomposing Proof Terms for String Rewriting into Multistep Derivations by Topological Multisorting

Vincent van Oostrom[*]
University of Bath
Bath, England
vvo21@bath.ac.uk

We show that a proof term in a string rewrite system can be mapped to its causal graph and that the latter is a unique representative of the permutation equivalence class of the former. We then map the causal graph back to a proof term of a special shape, a so-called greedy multistep reduction. Composing both transformations yields a simple and effective way of constructing the greedy multistep reduction of a proof term, and thereby of deciding permutation equivalence of proof terms in general, and of (multistep) reductions in particular.

1 Introduction

In general, we are interested in all aspects of computations as modelled by rewrite systems. Here, we are interested in *finite* computations 'doing the same work up to the order of the tasks performed'. This can be analysed from the perspective of *causality* [21, 8] with the idea that it is exactly the causally independent tasks that can be reordered. In term rewriting the ensuing notion of equivalence of computations is known as *permutation* equivalence and is well-studied [12, 10, 15, 13, 2, 6]. Our vantage point here is [22, Section 8.1.3] *Causal equivalence*. The goal of this short note then is to exhibit the close ties between the theory of permutation equivalence and developments in neighboring areas in [24, 25] (physics) and [4] (algebra). Concretely, this note was provoked by a remark in 2020 by Jan Willem Klop that Wolfram's causal graphs should characterise permutation equivalence. Being aware of that already, my short reply was to refer him to [22]. That was too cryptic. This is the long reply (whose essence is in Figures 1, 2).

The characterisation presented here is for *string* rewriting and hinges on two ideas. The first is to represent computations in rewrite systems as terms themselves, so-called *proof terms* [15, 22]. Consequently, computations may be given meaning in an *algebraic* way, namely via so-called *proof term algebras* for their representing proof terms. Each such proof term algebra induces a notion of equivalence of computations, namely, computations having the same interpretation are deemed equivalent. The subsequent idea is to employ a proof term algebra exactly capturing *permutation* equivalence. We show that Wolfram's *causal graphs* serve that purpose, and arise from a proof term algebra whose objects we call *tragrs* (pronounce as *trackers*), short for *trace graphs*, which are a variation on the *internal trace relation* of [22, Definition 8.6.17]. Finally, we show that *topological (multi)sorting* allows to map any tragr back to the *unique* computation that is both permutation equivalent to the original one and *greedy* in the sense of [4], i.e., in which tasks are performed as early as possible as allowed by causality.

Referring the reader to [15, 22] for formal background on proof terms, here we only give their idea and illustrate their versatility by two examples from category theory and deep inference. The basic idea is to extend the signature with symbols for *rules*. A term over such an extended signature is naturally interpreted as a *multistep* whose *source* / *target* is obtained by (homomorphically) mapping each rule in

[*]Supported by EPSRC Project EP/R029121/1 Typed lambda-calculi with sharing and unsharing.

Submitted to:
TERMGRAPH 2022

it to its left-hand side / right-hand side. *Proof terms* are obtained from this by further adding a symbol $\cdot$ to represent *composition* (respecting sources and targets) in the mix. A proof term can thus be seen as witnessing statements (of rewrite logic [15]) of shape $s \geqslant_P t$ expressing that the (target) string t is *reachable* from the (source) string s using rules from P.

Example 1. 1. *Consider the term rewrite system (TRS) expressing associativity of the operation @:*

$$\alpha(x,y,z) \quad : \quad @(@(x,y),z) \quad \to \quad @(x,@(y,z))$$

with α a (ternary) rule symbol. The TRS is terminating, so to show confluence it suffices by Huet's critical pair lemma to check its critical peaks are joinable. Here there's only one such, with source $@(@(@(w,x),y),z)$. Its local confluence diagram may be rendered using proof terms, as:

$$@(\alpha(w,x,y),z)\cdot\alpha(w,@(x,y),z)\cdot @(w,\alpha(x,y,z)) \quad \Rightarrow \quad \alpha(@(w,x),y,z)\cdot\alpha(w,x,@(y,z))$$

Both sides are proof terms: compositions (expressed by $\cdot$) of steps (expressed by terms over @ and *α). For example, $\alpha(w,@(x,y),z)$ denotes a step having source $@(@(w,@(x,y)),z)$ and target $@(w,@(@(x,y),z))$. The diagram captures* exactly *the* pentagon *diagram of monoidal categories.*

2. *Consider the TRS expressing inference rules medial* m *and contraction* $\mathsf{ac}{\downarrow}_a$*, for each constant a:*

$$\begin{array}{rcrcl} \mathsf{m}(w,x,y,z) & : & (w\wedge x)\vee(y\wedge z) & \to & (w\vee y)\wedge(x\vee z) \\ \mathsf{ac}{\downarrow}_a & : & a\vee a & \to & a \end{array}$$

Combining the idea of deduction as reduction *of [9, Section 4.4] with representing reductions as proof terms as above, allows to express the two* SKS*-deductions of [5, Example 2.15]* exactly *as:*

$$(\mathsf{m}(a\wedge b,c,a\wedge b,c))\cdot(((a\wedge b)\vee(a\wedge b))\wedge\mathsf{ac}{\downarrow}_c)\cdot(\mathsf{m}(a,b,a,b)\wedge c)\cdot((\mathsf{ac}{\downarrow}_a\wedge(b\vee b))\wedge c)\cdot((a\wedge\mathsf{ac}{\downarrow}_b)\wedge c)$$

$$\mathsf{m}(a\wedge b,c,a\wedge b,c)\cdot((\mathsf{m}(a,b,a,b)\cdot(\mathsf{ac}{\downarrow}_a\wedge\mathsf{ac}{\downarrow}_b))\wedge\mathsf{ac}{\downarrow}_c$$

The latter being the canonical form *[22, Definition 8.3.4] of the former captures* exactly *that the latter's* synchronal SKS*-deduction is obtained by* synchronising *the former's* sequential *one [5, Example 2.19].*[1] *Both (d/r)eductions perform the same tasks so are permutation equivalent [22].*

Since permutation equivalence was defined for *term* rewriting in [22] and we consider *string* rewriting, we extend the 2 standard embeddings [22, Section 3.4.4] of the latter in the former, to proof terms.

One is (what one could call) the *oudenadic* embedding, viewing letters in the string alphabet as *nullary* symbols. In that embedding, strings are constructed from the letters by means of nullary (for empty string), and binary (for append) constructs (giving the free monoid). The drawback of that is that we must then work modulo the monoid identities for these to obtain unique representatives of strings.

The other is the *monadic* embedding, viewing the letters in the string alphabet as *unary* symbols. This embedding does away with the drawback of the oudenadic embedding by absorbing the monoid equations; strings are uniquely represented as terms. A drawback is that an asymmetric (top–down) structure of terms is imposed on the symmetric (left–right) structure of strings.

We first relate both embeddings, by translations, introducing proof terms and permutation equivalence for the oudenadic embedding such that these correspond to the extant notions of [22, Chapter 8].

[1] Technically, [5, Theorem 2.18] is an instance of [22, Exercise 8.3.6].

2 Embedding string rewriting into term rewriting

We consider term rewrite systems in the sense of [22, Chapters 8 and 9] meaning that rules themselves are symbols whose arity is the number of variables in (the left-hand side of) the rule, and rules come equipped with source / target functions mapping them to their lhs / rhs. This enables expressing proofs in rewrite logic as *proof* terms [15, 22], terms over a signature comprising the letters, the rules, and a binary composition symbol $\cdot$ representing the transitivity inference rule of rewrite logic [15].

Definition 1. *A term rewrite system is* oudenadic *if all rule and function symbols have arity* 0*, it has a nullary symbol ε (empty string) and a(n implicit) binary symbol (juxtaposition), and terms are considered modulo the monoid laws for the latter, i.e. $\varepsilon s = s$, $s\varepsilon = s$, and $(st)u = s(tu)$.*

A term rewrite system is monadic *if all rule and function symbols have arity* 1 *and each rewrite rule $\rho(\varepsilon) : \ell \to r$ is* linear*, meaning in this case that the source ℓ and target r have the same free variable, ε.*

Identifying a string with an oudenadic term modulo the monoid laws, and a string rewrite system with its oudenadic term rewrite system modulo the monoid laws, we let $\sharp$ be the function mapping each equivalence class of strings modulo the monoid laws to a monadic term,[2] *and mapping the string rewrite system $\langle \Sigma, P\rangle$ to the monadic term rewrite system $\langle \Sigma^\sharp, P^\sharp\rangle$, where $\Sigma^\sharp$ is Σ with all symbols being assigned arity* 1*, and $P^\sharp$ having a rule $\rho^\sharp : \ell^\sharp \to r^\sharp$ for each rule $\rho : \ell \to r$ in P. We use $\flat$ to denote the inverse map from monadic to oudenadic term rewriting.*

We assume sources and targets of rules to be nonempty strings (see Remark 3). Our usage of the notation ε commonly used to denote the empty string, to denote an arbitrary but fixed free variable is justified by that the embedding maps the empty string to that free variable:

Example 2. *Consider the string $ABAAB$ in the string rewrite system $\langle \Sigma, P\rangle$ with alphabet $\Sigma := \{A, B\}$ and rules $P := \{\alpha : BB \to A, \beta : AAB \to BAAB\}$. The corresponding monadic term is $A(B(A(A(B(\varepsilon)))))$ and the corresponding monadic term rewrite system has a signature embedding the letters $\{A, B\}$ as unary function symbols, and the rules as $\alpha(\varepsilon) : B(B(\varepsilon)) \to A(\varepsilon)$ and $\beta(\varepsilon) : A(A(B(\varepsilon))) \to B(A(A(B(\varepsilon))))$.*

In line with naming the functions $\flat$ / $\sharp$ we refer to oudenadic / monadic rewrite systems as *flat* / *sharp*. The functions $\flat$ and $\sharp$ are inverse to each other. We extend this to proof terms [15, 22].

(empty)	$\varepsilon :$	$\varepsilon \geqslant^\flat \varepsilon$		(left unit)	$s \cdot \gamma \equiv^\flat \gamma$
(letter)	$a :$	$a \geqslant^\flat a$	for each letter a	(right unit)	$\gamma \cdot t \equiv^\flat \gamma$
(rule)	$\rho :$	$\ell \geqslant^\flat r$	for each rule $\rho : \ell \to r$	(associativity)	$(\gamma \cdot \delta) \cdot \zeta \equiv^\flat \gamma \cdot (\delta \cdot \zeta)$
(juxtaposition)	$\gamma_1\gamma_2 :$	$s_1s_2 \geqslant^\flat t_1t_2$	if $\gamma_i : s_i \geqslant^\flat t_i$	(exchange)	$\gamma\delta \cdot \zeta\eta \equiv^\flat (\gamma \cdot \zeta)(\delta \cdot \eta)$
(transitivity)	$\gamma \cdot \delta :$	$s \geqslant^\flat u$	if $\gamma : s \geqslant^\flat t$ and $\delta : t \geqslant^\flat u$		

Table 1: Proof terms for string rewriting (left) and their permutation equivalence (right)

Definition 2. Proof *terms for an oudenadic system are an inductively defined subset of the terms over the signature of the system extended with the rules ρ in P as nullary symbols, and a binary composition $\cdot$ (representing transitivity). They come equipped with* source *and* target *functions* src *and* tgt *mapping them to strings. We use $\gamma : s \geqslant^\flat t$ to denote that γ is a proof term having the strings s and t as source*

[2] This itself can be achieved by term rewriting: orienting the monoid laws from left to right yields a complete (confluent and terminating) term rewrite system, having as normal forms strings of shape either ε or $a_1 \ldots a_n$ with juxtapositions associated to the right; such a string is mapped to the monadic term ε respectively $a_1(\ldots(a_n(\varepsilon)\ldots)$.

and target. The clauses inductively defining proof terms are given on the left in Table 1. Juxtaposition is again taken modulo the monoid laws (this is compatible with sources and targets), and is assumed to bind stronger than $\cdot$. Proof *terms for a monadic system are defined analogously, see [22, Definition 8.2.18].*

We employ $\gamma,\delta,\zeta,\eta,\ldots$ to range over proof terms of either system. To keep the respective *reachability* relations apart we use $\geqslant^\flat$ for the former and $\geqslant^\sharp$ for the latter.

Example 3. *For the string rewrite system of Example 2, the proof term* $\gamma := AB\beta\cdot A\alpha AAB\cdot AA\beta\cdot\beta AAB\cdot B\beta AAB\cdot\alpha AABAAB\cdot A\beta AAB$ *witnesses that* $ABAAB \geqslant^\flat ABAABAAB$. *An alternative, shorter, witness to the same reachability statement is the proof term* $\gamma' := AB\beta\cdot A\alpha\beta\cdot\beta AAB\cdot B\beta AAB\cdot\alpha\beta AAB$.

(empty)	$\varepsilon^\sharp := \varepsilon$	(empty$^\sharp$)	$\varepsilon^\flat := \varepsilon$
(letter)	$a^\sharp := a(\varepsilon)$	(letter$^\sharp$)	$(a(\gamma))^\flat := a\gamma^\flat$
(rule)	$\rho^\sharp := \rho(\varepsilon)$	(rule$^\sharp$)	$(\rho(\gamma))^\flat := \rho\gamma^\flat$
(juxtaposition)	$(\gamma_1\gamma_2)^\sharp := s_1^\sharp(\gamma_2^\sharp)\cdot\gamma_1^\sharp(t_2^\sharp)$		
(transitivity)	$(\gamma\cdot\delta)^\sharp := \gamma^\sharp\cdot\delta^\sharp$	(transitivity$^\sharp$)	$(\gamma\cdot\delta)^\flat := \gamma^\flat\cdot\delta^\flat$

Table 2: Translations $\sharp$ from flat proof terms to sharp ones (left) and vice versa $\flat$ (right)

Lemma 1. $s \geqslant^\flat t$ *iff* $s^\sharp \geqslant^\sharp t^\sharp$ *by extending* $\sharp$ *and* $\flat$ *to flat and sharp proof terms as per Table 2.*[3]

Remark 1. *The intuition that the flat system is left–right symmetric but the sharp system is not, is captured formally by that although the sharp system may accommodate context / prefix-sharing as in* $a(\rho(\varepsilon)\cdot\vartheta(\varepsilon)) : a(a(\varepsilon)) \geqslant^\sharp a(c(\varepsilon))$ *for string rewrite rules* $\rho : a \to b$ *and* $\vartheta : b \to c$, *suffix-sharing as in the flat* $(\rho\cdot\vartheta)a : aa \geqslant^\flat ca$ *is impossible in the sharp system, and translated away by the sharp translation, yielding* $\rho(a(\varepsilon))\cdot\vartheta(a(\varepsilon))$ *('decorated' with empty steps*[3]*) in which the suffix a indeed is 'unshared'.*

The flat translation $\rho a\cdot\vartheta a$ of the sharp term $\rho(a(\varepsilon))\cdot\vartheta(a(\varepsilon))$ in the remark is distinct from the original proof term $(\rho\cdot\vartheta)a$. That is, although by Lemma 1 the flat and sharp translations preserve reachability, they are typically not inverse to each other. Still, the translations are natural in that rule applications are preserved, so that proof terms 'do the same work'. The latter is captured by the notion of *permutation* equivalence $\equiv$ between proof terms [12]. For strings, permutation equivalence $\equiv^\flat$ is generated by the equivalences on the right in Table 1, and for terms permutation equivalence $\equiv^\sharp$ is generated by the equivalences in [22, Table 8.2]. The next lemma entails that the translations in Lemma 1 between flat and sharp proof terms, although not inverse to each other, *are* so *modulo* permutation equivalence.

Lemma 2. *If* $\gamma \equiv^\flat \delta$ *then* $\gamma^\sharp \equiv^\sharp \delta^\sharp$. *If* $\gamma \equiv^\sharp \delta$ *then* $\gamma^\flat \equiv^\flat \delta^\flat$. $\gamma \equiv^\flat (\gamma^\sharp)^\flat$ *and* $\gamma \equiv^\sharp (\gamma^\flat)^\sharp$.

The above justifies freely switch between flat and sharp proof terms as we do below. E.g., we may just write $\geqslant$ and $\equiv$. Whereas the proof terms in Remark 1 exhibit sharing, those in Example 3 don't.

Definition 3. *A* multistep *is a proof term without occurrences of* $\cdot$. *It is* empty */ a (*single*)* step *if it has no / one occurrence of a rule. A (multi)step* reduction *either is an empty multistep or a composition of nonempty (multi)steps.*

Multisteps may alternatively be defined as strings over the letters Σ and the rules P. We use $\Phi,\Psi,X,\ldots$ to range over multisteps, and $\phi,\psi,\chi,\ldots$ to range over steps. If t is reachable from s, then this may be witnessed by a (multi)step reduction, known as *logicality* of (multistep) reduction for rewrite logic [17]. This can be brought about in general, in a permutation equivalence preserving way:

[3]The inside–out order chosen in (juxtaposition) corresponds to the one in deep inference, but is in principle arbitrary; any sequentialisation will do. Although ordering is indeed not compatible with working modulo the monoid laws, as pointed out by one of the reviewers, $\sharp$ can be defined on representatives as per footnote 2.

Lemma 3 (cf. [15, Lemma 3.6])**.** *For every flat proof term γ, there is a (multi)step reduction $\gamma' \equiv \gamma$.*

The lemma expresses that every proof term *factorises* as the vertical composition of a number of horizontal juxtapositions (see Figure 2), i.e. as the sequential composition of parallel compositions of steps. Composing the sharp and flat translations as in Lemma 2 does even more, *sequentialising* the steps *contained* in a parallel composition into a sequence of single steps; it favours length over width so to speak. In the next sections, we will go into the opposite direction, maximally favouring width over length (*area* being an invariant). For instance, the proof term γ in Example 3 is wasteful in two ways:

(too long) This can be remedied by proceeding greedily [4], employing proper multisteps instead of steps. For instance, the second and third steps $A\alpha AAB \cdot AA\beta : ABBAAB \geqslant AABAAB$ in γ can be combined into the single multistep $A\alpha\beta : ABBAAB \geqslant AABAAB$. Proceeding greedily, combining as many of the single steps into multisteps as possible, and as early as possible, turns γ into the shorter *greedy* multistep reduction γ' given in the same example.

(too large) This still holds true for γ'. The point is that (multi)steps not only represent what changes, rules, but also what *does not change*, letters (cf. the frame problem). As a consequence, multisteps predominantly consist of letters. Causal graphs [24] (as in Figure 1 below) remedy this.

To express both remedies we will employ a bit of *residual* theory for multisteps (going back to [3] for multisteps for the β-rule). To avoid things becoming too heavy for this short note, we only develop the residual theory necessary here and in an ad hoc informal fashion, referring the reader to Chapter 8 of [22] in general and to Section 8.7 in particular, for background on (from the perspective of permutation equivalence), and a formal treatment of, residuation.

Definition 4. *For multisteps Φ, Ψ having the same source, we write $\Phi \subseteq \Psi$ to denote that Φ is* contained *in Ψ, meaning that Φ is obtained from Ψ by mapping* some *occurrences of rules to their source. In that case, we denote by Ψ/Φ the* residual *of Ψ* after *Φ, that is, the multistep obtained from Ψ by mapping the* other *occurrences of rules (the* complement *of those selected for $\Phi \subseteq \Psi$) to their target.*

For instance, $ABBAAB$, $ABB\beta$, $A\alpha AAB$ and $A\alpha\beta$ are the four multisteps contained in $A\alpha\beta$ in Example 3. We have, e.g., $A\alpha\beta/ABB\beta = A\alpha BAAB$ and $A\alpha\beta/A\alpha AAB = AA\beta$. Observe that if $\Phi \subseteq \Psi$ and Φ is nonempty, then fewer rules occur in Ψ/Φ than in Ψ.

3 Greedy multistep reductions

We give a standard algorithm for transforming a multistep reduction into a greedy one [4], but using a novel description of greediness and the greedy algorithm, based on the analogy with sorting and standardisation [10, 22] in the literature. In sorting, (adjacent) *inversions* are consecutive elements that are out-of-order, and in term rewriting, *anti-standard* pairs [10, 14] are consecutive steps in a reduction such that the latter is outside (to the left of) the former. Such pairs of out-of-order elements are of interest since they provide a *local* characterisation both of *being sorted*, i.e. the *absence* of such pairs, and of bringing the list / reduction *closer to being sorted*, by *permuting* the out-of-order pair. This makes both processes amenable to a rewriting approach, with bubblesort being an example for sorting and the extraction of the leftmost-contracted-redex being an example for standardisation [10, 7, 13, 22, 14, 2]. To make the greedy algorithm fit the same mould, we identify *loath*, i.e. *non*-greedy, pairs of multisteps.

Definition 5. *A pair of consecutive multisteps Φ, Ψ is (X-)*loath *if X is a* witnessing *multistep such that $\Phi \subseteq X$ with $X/\Phi \subseteq \Psi$ nonempty (with X/Φ nonempty and $X/\Phi \subseteq \Psi$, that is). Its (X-)*swap *is defined to be the pair $X, \Psi/(X/\Phi)$. A* greedy *reduction is one without loath pairs.*

Lemma 4. *If* Φ,Ψ *is* X-loath*, then* $\Phi\cdot\Psi \equiv X\cdot(\Psi/(X/\Phi))$.

Setting X to the patterns in Φ and *all* those in Ψ that 'do not overlap Φ', minimises swapping.

Example 4. *The proof term γ in Example 3 comprises two loath pairs. Working from the tail to the head, we first encounter the loath pair $\alpha AABAAB, A\beta AAB$ between the the 6th and 7th steps, which is permuted into $\alpha\beta AAB, ABAABAAB$. The next loath pair $A\alpha AAB, AA\beta$ is between the 2nd and 3rd steps and is permuted into $A\alpha\beta, AABAAB$. After this, we proceed by only permuting loath pairs that are trivial, in the sense that their first multistep is empty (permuted into pairs where the second multistep is empty), resulting in $\gamma'\cdot ABAABAAB\cdot ABAABAAB$, for γ' as in Example 3.*

Instead of permuting trivial loath pairs, which is not useful as the example shows, we will instead implicitly elide all empty multisteps, so that Example 4, starting from γ we indeed end up with the greedy γ', not with γ' followed by two empty multisteps.

Lemma 5. *Repeated swapping terminates in a permutation equivalent greedy multistep reduction.*

In the next section we show that by suitably *evaluating* a proof term, we may obtain the greedy multistep reduction directly and as a consequence the latter is *unique* modulo permutation equivalence.

Remark 2. *An efficient procedure for searching for loath pairs can be based on the observation that due to linearity of string rewrite systems, an occurrence of either a source or target of a rule can be identified with a* pattern *in the sense of [22, Definition 8.6.21], i.e. with a* convex *set of positions in the tree of the string having vertices as boundary. Following the main idea of [18], to see whether Φ,Ψ is loath, it therefore suffices to check whether each pattern of a* source *of a rule occurring in Ψ has overlap with some* target *of a rule occurring in Φ. Since a pattern in a string simply is an* interval*, characterised by the two vertices constituting its boundary, a single top–down pass through both string-trees checking disjointness of intervals via their boundaries, suffices. If for some pattern there is no overlap, we obtain a loath pair by setting X to Φ in which the pattern was replaced by the rule.*

For example, using underlining to indicate occurrences of patterns, that $\alpha AABAAB, A\beta AAB$ in γ is a loath pair follows from that the pattern $\{\mathring{2},\overline{3},\mathring{3},\overline{4},\mathring{4}\}$ in $A\underline{AAB}AAB$ corresponding to the source AAB of the rule β, does not have overlap with the pattern $\{\mathring{1}\}$ in $\underline{A}AABAAB$ corresponding to the target A of the rule α. This in turn follows from that the corresponding intervals $[\mathring{2},\mathring{4}]$ and $[\mathring{1}]$ are disjoint since $\mathring{1}$ is smaller than $\mathring{2}$. By disjointness / non-overlap, replacing in $\alpha\underline{AAB}AAB$ the β-pattern AAB by the rule β yields the multistep $\alpha\beta AAB$, as desired.

4 Causal graphs by proof term algebra

We give a proof term algebra such that evaluating it for a proof term yields its causal graph [24]. A proof term algebra for a string rewrite system $\langle\Sigma,P\rangle$ is like an ordinary term algebra, but for the signature $\Sigma\uplus P$ comprising both the letters Σ and the rules P. Moreover the algebra should cohere with the functions src and tgt mapping a proof term to its source / target string [15]. *Evolution* [24] is a proof term algebra:

Example 5. *The goal is to have the proof term γ of Example 3 evaluate to the top segment (the first 15 rows) of the evolution on the left of in Fig. 1 (taken from [20]; based on [24, fig. a, p.498]). To that end, we interpret multisteps as rows of (possibly skewed) blocks obtained by $A\mapsto$ □, $B\mapsto$ ■, $\alpha\mapsto$ ▰, and $\beta\mapsto$ ▰. Compositions of multisteps are interpreted by stacking the rows of the interpretations of the multisteps on top of each other, interspersed with the evaluations of their sources and targets. For instance, the top three rows are the evaluations □■□□■, □■▰, and □■■□□■ of the source, multistep, and target of $AB\beta : ABAAB \geqslant ABBAAB$. (The interpretations of α,β are next to the evolution).*

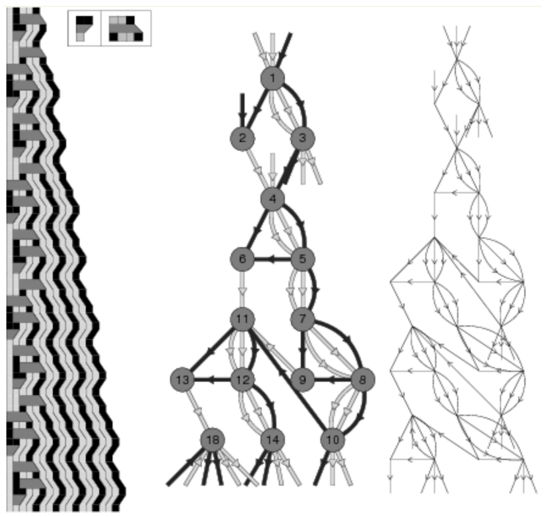

Figure 1: Evolution and causal network of *ABAAB* for string rewrite system $BB \to A$, $AAB \to BAAB$ [20]

We show that the *causal graphs* of [24] can be obtained by means of another proof term algebra capturing *causal equivalence* as in [22, Section 8.1.3], using a graph to represent what *causes* each occurrence of a rule in a proof term, how these are caused by letters in the source and by other rules in the proof term. For this to work rules must *cause*, i.e. their source / target nonempty.

Remark 3. *Empty sources and targets are conceptually and technically problematic. Conceptually, they allow infinite outward (ex nihilo instead of ex materia steps) respectively inward branching and it is not clear whether that meshes well with causality, cf. [23]. Technically, empty targets are problematic for causality via causal graphs as witnessed by rules* $\rho : c \to \varepsilon$ *and* $\vartheta : aa \to b$. *Then* $a\rho a \cdot \vartheta : aca \geqslant b$, *but the steps cannot be put into the same multistep since clearly the second is created by the first. However, in the causal graph neither of the two a-symbols constituting the lhs of* ϑ *is caused by the* ρ *because that has an empty rhs. As far as we know, this is not discussed in [24]; see [22, Remark 8.6.75] or the Remark below Example 3 in [16] for two known work-arounds (*reifying edges / splitting symbols*).*

The goal is for proof terms to have the same evaluation in the algebra iff they are causally equivalent. For string rewriting, this is *almost* achieved by the proof term algebra of [22, Definition 8.6.17] based on *trace relations* (see [22, Figure 8.37] for an illustration; trace relations are known as *atomic flows* in deep inference as in Example 1.2.) We exemplify that proof term algebra, mainly to exhibit what is still missing to *exactly* capture causal equivalence.

Example 6. *Trace relations are relations between the positions in (the term tree of) proof terms. The result of computing the* source s, internal i, *and* target t *trace relations [22, Def. 8.6.17] for the proof term* γ *of Example 3, its source ABAAB, and its target ABAABAAB is depicted on the left in Figure 2, where we have written the strings constituting* γ *(rather than the positions of the symbols within* γ*) horizontally, and used dotted lines to indicate the induced trace relations. All 'edges' are directed downward.*

The defect of proof terms being (too large) mentioned above, still applies to (internal) trace relations as they have the set of positions of a proof term as domain. But observe letters in Σ may be elided since they do not change causality: each letter relates and is related to exactly one other position (they represent identities; cf. the text below [22, Def. 8.6.17]). Doing so for γ results in the trace relation in the middle in Figure 2. That trace relation is identical to the initial segment (the first 7 steps; with the correspondence

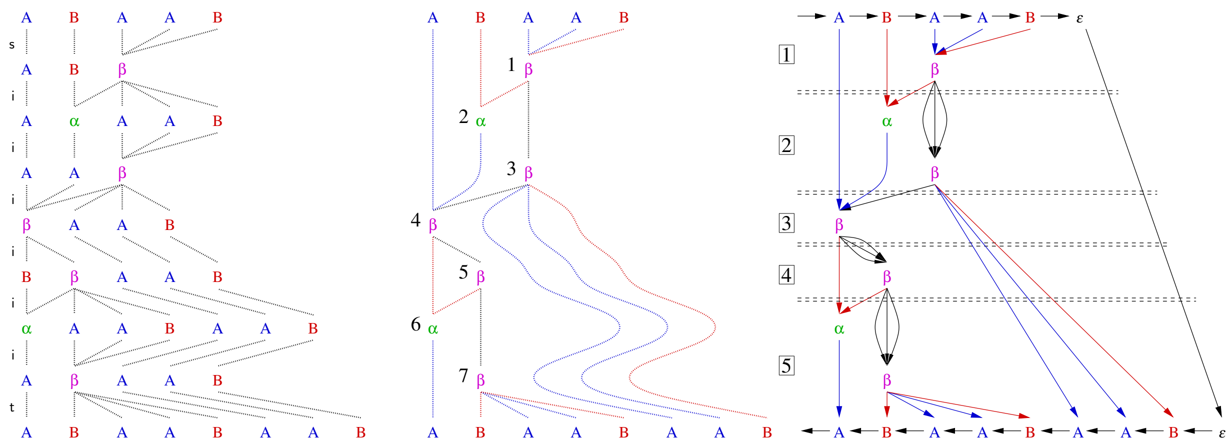

Figure 2: Trace relations and graph of $(AB\beta)\cdot(A\alpha AAB)\cdot(AA\beta)\cdot(\beta AAB)\cdot(B\beta AAB)\cdot(\alpha AABAAB)\cdot(A\beta AAB)$

indicated by the numbers) of the *causal graph* in the middle of Figure 1, except for *parallel* edges in the latter. To regain these as well, we change to evaluating into trace *graphs* instead of trace *relations*.

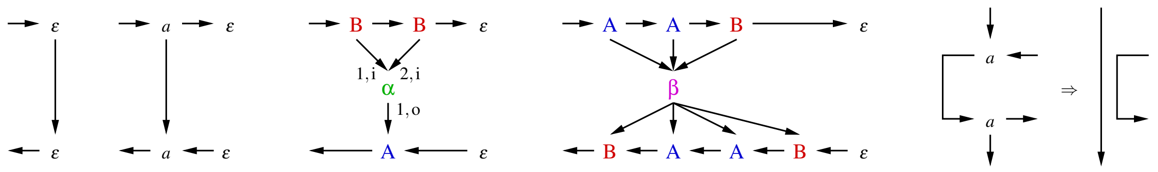

Figure 3: Trace graphs for ε, letter a, rules α and β, and elision rule $\Rightarrow$ for letter a or ε

Definition 6. *The* trace graph *(*tragr*) proof term algebra is given (by example) in Figure 3. It (recursively) evaluates a proof term $\gamma : s \geqslant t$ as a graph denoted by $[\![\gamma]\!]$ having unique input and output edges (on the left), with nodes labelled by symbols in $\Sigma \cup P \cup \{\varepsilon\}$, with labels determining the* ports *of a node, cf. [1, 19]. For each occurrence of a letter in the source s (the target t) there is a correspondingly labelled node having one input port, one output port, and a down (up) port, and these nodes are chained forward ending in (backward starting from) an ε-node; the chain is called the* source / target *dag. The number of input / output ports of a rule is determined by the length of its source / target string. We name ports of rules by their index and a boolean 'i' / 'o' as shown for α in Figure 3 (only done for α there, since for β it would lead to clutter).*

- *The tragr $[\![\gamma\delta]\!]$ is the juxtaposition of the tragrs $[\![\gamma]\!]$ and $[\![\delta]\!]$ redirecting the edge to (from) the top (bottom) ε-node in the former to the input (output) edge of the latter; and*
- *The tragr $[\![\gamma \cdot \delta]\!]$ is the composition of the tragrs $[\![\gamma]\!]$ and $[\![\delta]\!]$, connecting the output of the former to the input of the latter, and then eliding the intermediate target and source dags (for $\mathtt{tgt}(\gamma)$ and $\mathsf{src}(\delta)$), by normalising with respect to the* elision *rule $\Rightarrow$ displayed on the right in Figure 3.*

An example tragr is displayed on the right in Figure 2. Note that the tragr does reflect also the parallel edges of the causal graph in Figure 1, which were lacking in trace relations.

Remark 4. *Tragrs are acyclic and planar since the base cases are and both properties are preserved by the 'parallel' (juxtaposition) and 'series' (composition) operations.*

Elision is complete: terminating because the number of nodes decreases and confluent because $\Rrightarrow$ can be viewed as an interaction net rule [11].

Contrary to causal graphs, but in line with trace relations, each tragr has a source / target dag, a dag-representation of the source / target string of the proof term; the *interface* of the tragr. We consider this discrepancy minor and suggest that if the causal graphs of [24] were to be completely formalised, that would result in tragrs. Therefore, we consider them to be interchangeable and work with tragrs.

Lemma 6. $[\![\,]\!]$ *maps proof terms that are equivalent modulo permutation equivalence (Table 1), to the same tragr (up to isomorphism).*

Theorem 1. *There is a(n effective) bijection between greedy multistep reductions and trace graphs.*

Proof idea. We give a function TS from tragrs to greedy multistep reductions performing a *topological* (multi)*sort* of a tragr by selecting at each stage *all* minimal nodes, and show that TS and $[\![\,]\!]$ (restricted to greedy multistep reductions) are inverse to each other. Executing TS on the tragr on the right in Figure 2 yields stages $\boxed{1}$ to $\boxed{5}$ (giving rise to 5 multisteps) as separated by double–dashed lines. □

Using the above, we establish our main result that one may compute a greedy multistep reduction, unique modulo permutation equivalence, for any proof term by first evaluating into its tragr / causal graph (using $[\![\,]\!]$), followed by the topological multisort (using TS) yielding the greedy multistep reduction. To that end, we employ a simple case of Hardin's interpretation method we dub *confluence-by-evaluation*.

Lemma 7 (CbE)**.** *Let* nf *be a function on the objects of a rewrite system* $\to$ *such that if* $a \to b$ *then* $\mathsf{nf}(a) = \mathsf{nf}(b)$*, and if* a *is a normal form, then* $\mathsf{nf}(a) = a$*. Then* $\to$ *is confluent, if* $\to$ *is normalising (WN).*

Proof. Suppose $b \twoheadleftarrow a \twoheadrightarrow c$. By $\to$ being normalising applied to b,c, there exist normal forms b',c' such that $b' \twoheadleftarrow b \twoheadleftarrow a \twoheadrightarrow c \twoheadrightarrow c'$. By the assumptions and convertibility of b',c', $\mathsf{nf}(b') = \mathsf{nf}(c')$, so $b' = c'$. □

Theorem 2. *For every proof term* γ*, there is a unique greedy multistep reduction* γ' *such that* $\gamma \equiv \gamma'$*.*

Proof. Instantiating $\to$ in Lemma 7 with the swap relation on multistep reductions and nf with the composition of $[\![\,]\!]$ with TS, we have that its assumptions are satisfied: $\to$ is (even strongly) normalising by Lemma 5, nf is the identity on greedy normal forms by Theorem 1, and nf maps permutation equivalent proof terms to the same greedy normal form since $[\![\,]\!]$ does so by Lemma 6, so nf is certainly preserved by swap steps since these preserve permutation equivalence by Lemma 4. Therefore, swapping is both terminating and confluent, i.e. complete. We conclude noting that, by Lemma 3, any proof term can be transformed into a multistep reduction that is permutation equivalent. □

Remark 5. *We call the method employed confluence-by-evaluation (CbE) as it only relies on semantics. It does* not *rely on Newman's Lemma and the combination of termination (Lemma 5) and local confluence of swapping (which we didn't show, although one could). For another example of CbE: that the TRS for multiplication and addition of [22, Table 2.3] is (ground) confluent follows from that it is WN and that the natural numbers constitute a model and are in bijection with the (constructor) normal forms* $S^n(0)$*.*

References

[1] A. Bawden (1986): *Connection Graphs*. In: *LFP 1986*, ACM, pp. 258–265, doi:10.1145/319838.319868.

[2] H.J.S. Bruggink (2008): *Equivalence of Reductions in Higher-Order Rewriting*. Ph.D. thesis, Utrecht University. Available at `http://dspace.library.uu.nl/handle/1874/27575`.

[3] A. Church & J.B. Rosser (1936): *Some properties of conversion*. Transactions of the American Mathematical Society 39, pp. 472–482, doi:10.1090/S0002-9947-1936-1501858-0.

[4] P. Dehornoy & alii (2015): *Foundations of Garside Theory*. European Mathematical Society, doi:10.4171/139.

[5] A. Guglielmi, T. Gundersen & M. Parigot (2010): *A Proof Calculus Which Reduces Syntactic Bureaucracy*. In: RTA 2010, LIPIcs 6, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 135–150, doi:10.4230/LIPIcs.RTA.2010.135.

[6] T. Hirschowitz (2013): *Cartesian closed 2-categories and permutation equivalence in higher-order rewriting*. LMCS 9(3), doi:10.2168/LMCS-9(3:10)2013.

[7] Z. Khasidashvili & J.R.W. Glauert (1996): *Discrete Normalization and Standardization in Deterministic Residual Structures*. In: ALP'96, LNCS 1139, Springer, pp. 135–149, doi:10.1007/3-540-61735-3_9.

[8] Z. Khasidashvili & J.R.W. Glauert (2002): *Relating conflict-free stable transition and event models via redex families*. TCS 286(1), pp. 65–95, doi:10.1016/S0304-3975(01)00235-3.

[9] Z. Khasidashvili & V. van Oostrom (1995): *Context-sensitive conditional expression reduction systems*. ENTCS 2, pp. 167–176, doi:10.1016/S1571-0661(05)80193-8.

[10] J.W. Klop (1980): *Combinatory Reduction Systems*. Ph.D. thesis, Rijksuniversiteit Utrecht.

[11] Y. Lafont (1990): *Interaction Nets*. In: 17th POPL, ACM Press, pp. 95–108, doi:10.1145/96709.96718.

[12] J.-J. Lévy (1978): *Réductions correctes et optimales dans le λ-calcul*. Thèse de doctorat d'état, Université Paris VII. Available at `http://pauillac.inria.fr/~levy/pubs/78phd.pdf`.

[13] P.-A. Melliès (1996): *Description Abstraite des Systèmes de Réécriture*. Thèse de doctorat, Université Paris VII. Available at `http://www.irif.fr/~mellies/phd-mellies.pdf`.

[14] P.-A. Melliès (2005): *Axiomatic Rewriting Theory I: A Diagrammatic Standardization Theorem*. In: Essays Dedicated to Jan Willem Klop, LNCS 3838, Springer, pp. 554–638, doi:10.1007/11601548_23.

[15] J. Meseguer (1992): *Conditional rewriting logic as a unified model of concurrency*. Theoretical Computer Science 96, pp. 73–155, doi:10.1016/0304-3975(92)90182-F.

[16] V. van Oostrom (1997): *Finite Family Developments*. In: RTA-97, LNCS 1232, Springer, pp. 308–322, doi:10.1007/3-540-62950-5_80.

[17] V. van Oostrom (2004): *Sub-Birkhoff*. In: FLOPS 2004, LNCS 2998, Springer, pp. 180–195, doi:10.1007/978-3-540-24754-8_14.

[18] V. van Oostrom (2020): *Some symmetries of commutation diamonds*. In: IWC 2020, pp. 1–7. Available at `http://iwc2020.cic.unb.br/iwc2020_proceedings.pdf`.

[19] V. van Oostrom, K.J. van de Looij & M. Zwitserlood (2004): *Lambdascope*. In: ALPS 2004, p. 9. Available at `http://cl-informatik.uibk.ac.at/users/vincent/research/publication/pdf/lambdascope.pdf`.

[20] T. Rowland & E.W. Weisstein: *Causal Network*. Available at `https://mathworld.wolfram.com/CausalNetwork.html`.

[21] E.W. Stark (1989): *Concurrent Transition Systems*. TCS 64, pp. 221–269, doi:10.1016/0304-3975(89)90050-9.

[22] Terese (2003): *Term Rewriting Systems*. Cambridge University Press.

[23] G. Winskel (1989): *An introduction to event structures*. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, Springer, pp. 364–397, doi:10.1007/BFb0013026.

[24] S. Wolfram (2002): *A New Kind of Science*. Available at `https://www.wolframscience.com/nks/`.

[25] S. Wolfram (2020): *A Class of Models with the Potential to Represent Fundamental Physics*. Available at `https://www.wolframphysics.org/technical-introduction/`.